NTNU
Norwegian University of
Science and Technology

Deep Visual Domain Adaptation:

From Synthetic Data to the Real World

Magnus Reiersen

Master of Science in Computer Science
Submission date: August 2018
Supervisor: Theoharis Theoharis, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

In the field of computer vision, the increasing use of convolutional neural networks (CNN) fuels the need for more and more labeled training data. Synthetic data generated from computer graphics represent an alternative approach for fast acquisition of training data. However, synthetic data suffers from dataset bias, making models trained on synthetic data underperform "in-the-wild". In this master thesis, a survey comparing state-of-the-art domain adaptation techniques for CNNs in visual applications was conducted, accompanied by a brief look at how computer graphics can aid CNNs when real-world data is scarce. The survey concludes that many techniques are available for classification architectures, but the same principles used in classification can be used to extend architectures for other visual applications.
To further add to this research, several classical domain adaptation techniques consisting of different types of fine-tuning was attempted on the CNN architecture *Mask-RCNN*. The task was to predict salmon masks/silhouettes on photographs from real fish farms, but by pre-training on synthetic images of salmon in a virtual fish farming environment as a requisite. The synthetic pre-trained model managed 55.8 mAP(%) on synthetic images, but only 9.4 mAP(%) on real images, showing a dataset bias was present. To adapt to the real world, the pre-trained model was given only 19 fine-tuning real-world examples, making this a few-shot domain adaptation problem. The different fine-tuning techniques attempted was: *regular fine-tuning* and *gradually opening up layers for fine-tuning from a frozen state, starting from the deepest layers*. For both techniques it was further attempted to extend the small-real-word dataset by data augmentation. Real-world performance increased to 27.5% mAP after regular fine-tuning, 28.5 mAP(%) after gradually opening up layers in fine-tuning, and 41.9 mAP(%) and 36.5 mAP(%) mAP respectively with data augmentation. The regular fine-tuned model did 56.8 mAP(%) on synthetic images, showing that domain invariant features was learned. We argue that this must be due to a close overlap of distributions over computer graphics - and real-world images in the CNN solution-space. Furthermore, results also show that data augmentation can be used as supplement for extra performance on real-world images, but a dataset bias towards the real-world might become apparent.
As for gradually opening up layers, results suggests that this can help keep cross-domain performance, but overtraining on each stage can reduce the performance in-the-wild. However, more research is needed to support this claim.
Lastly, an unsupervised DA technique was attempted using a GAN trained for style-transfer to synthesize hybrid images (with labels taken from source). This failed and was likely due to the GAN not being designed for instance segmentation. This master thesis concludes that synthetic data used in a CNN will underperform compared to real-world data, but domain adaptation techniques can boost performance considerably such that synthetic data is as a good alternative to manual labeling training data.

Acknowledgements

I would like to acknowledge and thank the following important people and companies for supporting my research and work during the course of this master thesis.

Firstly, I would like to express my gratitude to my NTNU faculty supervisor, professor Theoharis Theoharis for guidance and support and setting the path straight for this thesis.

Secondly, I would like to thank Kongsberg Digital AS and Kongsberg Maritime AS for provision of data, equipment, guidance, and support. Here, I would like to give a special thanks to Pierluigi Salvo Rossi, Ph.D. for his support, follow-up, and guidance. He has been an important supervisor and intermediator between NTNU and Kongsberg Digital. Additionally, a special thanks to Christopher Dyken for providing me with code, API, technical support and documentation on Cogs and SimSalma. To all others helping me in Kongsberg Digital AS not mentioned her, I would also like to thank.

I would also like to thank Salmar ASA and Lerøy Midt AS for providing testing data from their facilities.

Preface

This master thesis was achieved in cooperation with Kongsberg Digital. My hope is that it will provide support to industry and academia when dealing with CNNs; that synthetic data combined with domain adaptation will break down the lines between the digital and the real, such that substantial work loads spent on harvesting data and labeling data will be shortened down. Writing this thesis has both been inspiring and challenging, as a lot of work was put into it, but at the same time I was given the opportunity to learn so much, in a field of technology that is on the verge of break-through. What an exciting time we are living in. In five years, many of the principles in this master is perhaps considered old news due to the fast progress in deep learning. One could perhaps think that this makes me less motivated, but rather not; having the opportunity to participate in such a revolutionary branch of science is truly inspiring.

Table of Contents

List of Figures

List of Tables

Nomenclature

DRCN	Deep Reconstruction-Classification Network, page 34
ADDA	Adversarial Discriminative Domain Adaptation, page 27
After Effects	An Adobe software developed for making visual effects for video industry, page 50
AlexNet	One of the first CNNs proposed for classification, containing simple conv. and pool operations , page 9
Autoencoder	Neural network set to learn efficient data codings in an unsupervised manner, page 33
BSW	Beyond Sharing Weights, page 31
CAD-models	Computer-aided design model, page 15
CNN	Convolutional neural network, page 2
CORAL	The squared Frobenius norm between the second order statistics (covariance) of two datasets, page 32
Covariate Shift	When the distribution of the inputs used as predictors (covariates) is shifted between two datasets, page 6
DA	Domain Adaptation, page 5
DANN	Domain-Adversarial Neural Network, page 25
DDC	Deep Domain Confusion, page 29
DSN	Domain Separation Network, page 34
Fast R-CNN	An CNN architecture built upon R-CNN designed to solve object detection, page 9

Faster R-CNN	A CNN architecture based on Fast-RCNN designed for solving the object detection problem, page 9
Feature Pyramid Networks	A type of CNN containing a pyramidal top-down architecture with lateral connections, page 11
Fine-tuning	Introducing more training data to a pre-trained network and continue training on the new training set, page 20
Freezing	Setting learning rate to some layers to 0, page 20
Gabor Filter	Linear filter used for texture analysis in computer vision, page 21
GAN	Generative Adversarial Network, page 23
GoogLeNet	A type of CNN containing inception modules for better conv. operations, page 11
IoU	Intersection of Union, page 80
JMMD	Joint Maximum Mean Discrepancy, page 33
mAP	Mean Average Precision, page 81
Mask-RCNN	An architecture building on Fast/Faster-RCNN set to solve the problem of instance segmentation, page 10
MMD	Maximum Mean Discrepancy, page 29
MNIST	Dataset containing pictures of handwritten digits, often used to test domain adaptation, page 19
Off-the-shelf	Chop off top layers on a CNN and train a machine learner using the network as a feature extractor on the new task, page 20
Office dataset	Dataset containing pictures of office objects, often used to test domain adaptation, page 19
PASCAL VOC	Provides standardised image data sets for object class recognition and run challenges, page 45
R-CNN	A method using classification CNNs multiple times on an image to turn it into an object detector, page 9
ResNet	A type of CNN using residual blocks to increase the amount of layers possible to train , page 79
ReverseGrad	Gradient Reversal layer, page 26
RKHS	Reproducing Kernel Hilbert Space, page 29

RNN	Recurrent Neural Network, page 3
RoI	Region of Interest, page 9
RoIAlign	Calculating a more precise RoI, page 11
RPN	Region Proposal Network, page 10
SimSalma	Kongsberg Digital's simulator for modeling salmon behavior, page 45
SSD	Single Shot MultiBox Detector, a CNN designed to solve object detection, page 39
Style-transfer	A technique to synthesize images based on the style of another image, page 23
SUNCG	Synthetic dataset contains 45,622 different scenes with realistic room and furniture layouts, page 16
SVHN	Another dataset containing pictures of handwritten digits, often used to test domain adaptation, page 19
SVHN	Dataset containing pictures of street sign digits, often used to test domain adaptation, page 19
SVM	Support Vector Machine, page 20
Unreal Engine 4	A computer graphics engine producing state-of-the-art realism, primarily used for game design , page 14
VGG16	A type of CNN used by the VGG team in the ILSVRC-2014 competition, page 11
YOLO	You Only Look Once, a CNN designed to solve object detection, page 39

Chapter 1

Introduction

1.1 Background of thesis

Kongsberg Gruppen have a three-year cooperative relationship with Ocean Farm 1, the deep sea marina farm. Here, Kongsberg will aid in monitoring the biomass of the farm, and explore how the fish is distributed in the feed volume. Also, the R&D program should be able to increase the attention on underwater situations and act as decision support for better control of feed operations. Using various sensors, big data will be collected on the whole facility, making Ocean Farm 1 a giant floating laboratory. At the same time, on land, two systems are developed to model the fish's external and internal status virtually, called SimSalma and CyberFish. These programs have the goal to aid research in modeling, predicting and controlling fish behavior. Specifically, SimSalma is a computer simulation, which can simultaneously model the behavior of hundreds of thousands of salmon inside a virtual farm. Much research and work have been put into making SimSalma closely modeling real-world fish farms, thus the software represents a method of easy access to big data resembling the real. Such data can hopefully aid to create new and better mathematical models in the future. Here, an interesting area of research is how synthetic data can be used to train various machine learning models set for later deployment in real-world situations. Many breakthroughs in machine learning have lately been attributed to the use of artificial neural networks. These networks have been praised for their performance and versatility on various applications. However, artificial neural networks, in general, requires a large amount of training data to generalize well beyond examples already seen in the training set. Using simulators such as SimSalma, a huge amount of data is easily accessible. The question remains if such models will perform well in real-world applications. In particular, how the synthetic quality affects the behavior of these networks is puzzling due to their "black box"-nature. Transferring between domains from the synthetic and the real represents an unknown jump where an outcome is difficult to predict.

1.2 Problem definition

The problem of this thesis can be defined in the following way:

Given a set of images X_s generated from 3D-computer graphics in a virtual domain D_s, that is; synthetic data with corresponding automatically generated labels Y_s, and a freely defined CNN architecture - a model M, dedicated to solve an arbitrary computer vision task, how can we deploy M with good performance on a target domain D_t represented by images X_t taken from the real world, when labeled data $\{X_t, Y_t\}$ for D_t is absent or few? With such methods in mind, can synthetic data from 3D-computer graphics be a favorable alternative to real data when used in CNNs?

This problem will be attempted answered in the entirety of this thesis, both theoretically and experimentally.

1.3 Objectives

In this thesis, we will look into domain adaptation for CNNs trained with synthetic data generated by 3D computer graphics, and see how different analytical and heuristic techniques are used in CNNs. We will look at various ways this could be done, as well as conduct an experiment using synthetic data generated by 3D computer graphics to attempt some techniques. Furthermore, the problem definition will be divided into the following tasks:

1. Investigate state-of-the-art deep domain adaptation methods for CNNs specifically set for visual applications.
2. Identify the behavior of computer graphics in domain adaptation
3. Take a CNN architecture and conduct domain adaptation on it using computer graphics as source training data and real photographs as target data. Here, we will focus on Mask-RCNN.
4. Theoretically discuss how this chosen CNN architecture can be modified in order to make domain adaptation on synthetic data easier.

From this, we can define some borders in our research topic. Since our objective is set for CNNs in particular, we can filter out topics on general domain adaptation and machine learning. However, since many domain adaptation ideas on CNNs are not explicitly bound to work only on CNNs, our study will still be relevant to most of the field of deep learning. Furthermore, domain adaptation techniques not able to perform on synthetic data by computer graphics (if such techniques exist) - we will also filter out. However, in practice, most techniques are not tested with various type of synthetic data, such that it will be assumed that all work more or less.

1.4 Structure of the Thesis

The thesis is structured into three parts mainly

1. Basic theory, synthetic data and DA literature study
2. Fine-tuning experiment
3. Results, conclusion and future work

The chapter on synthetic data (chapter 3) will work as an introduction, showing how computer graphics are being used in today's literature, and justifying why we need domain adaptation to answer our problem definition. During the literature study (chapter 4), we will explore state-of-the-art domain adaptation and its use in CNNs. The main emphasis of this literature study is domain adaptation. This means that our findings presented can be used for most visual applications in deep learning, and not just limited to cases with computer graphics, synthetic data and/or real images. Furthermore, since the task is set to be general for most of the CNN architectures, will try to take all we know from these findings and extend it towards other architectures in this chapter. Of these architectures, Mask-RCNN will be highlighted. However, Mask-RCNN is built up from other architectures, such that these architectures also needs to be explored. Moreover, all we need to know about these architectures and the domain adaptation problem will be presented in the chapter on *basic theory* 2. In chapter 5 (referring to (3) "Fine-tuning experiment)", an experiment on Mask-RCNN will be presented with domain adaptation techniques applied. We will here focus on *fine-tuning* due to its versatility, but many approaches to fine-tuning will be tested. During the experiment, synthetic data of salmon streams from SimSalma will be used to train a Mask-RCNN, solving the instance segmentation problem set on detecting and outlining salmon within a fish farming environment. Our target domain will be real data harvested from fish farms such as Ocean Farm 1 and one owned by Lerøy Midt. In the conclusion and future work section, we will wrap up all the knowledge collected, and try to answer our problem definition with multiple points.

1.5 Further restrictions

There is a lot of unexplored methods and concepts in the field of DA (domain adaptation) and on synthetic data. Unfortunately, due to time constraints will not have time to do a full review on everything relevant to this problem. For instance, domain adaptation is considered a comprehensive, not fully solved problem within computer vision, where a lot of research has been done, thus we will not have time to cover all of literature. Here we will only focus on the most well-known, state-of-the-art techniques. Moreover, there are many CNN, visual architectures, which have not been attempted with DA and should be discussed in relation to our topic. Here, we will cover some CNN architectures, but not all. One of the most influential branches within deep learning, which would be interesting to review in the context of DA is *recurrent neural networks* (RNN), which are brilliant for video application and other sequential visual application. However, these will not be covered.

Chapter 2

Basic Theory

Many concepts, definitions, principles and technologies important as a fundamental background for this master thesis will be described in this chapter. These are spread within different problems of computer vision, but will brought together at different times during the literature survey and experiment report. We will in this chapter first take a look at the nature behind the problem we are trying to address, domain adaptation and synthetic data for visual applications, then enlist some architectures relevant for our experiment.

2.1 Synthetic data

Synthetic data is defined as data meant to imitate some data taken from the real world. This means that various statistics of the synthetic data should match those of the real-world data. ”Real” is usually data taken from an environment where the agent harvesting such data must make passive observations, and can’t have direct control over it, e.g stock prices, weather information or user-data on a website. In the context of this thesis, this would be images from the real world, where we say that agents can’t have direct control over the physics of light behavior in nature. Synthetic images are usually generated from a 3D model using computer graphics which is set to imitate real light. The line between real and synthetic can be blurred, especially if real data is used to generate synthetic data such as style-transfer in GANs (more on this is in the next chapter). However, our problem is defined in the context of computer graphics vs real images.

2.2 Domain Adaptation and Transfer Learning

Within the literature, domain adaptation is considered a special case of transfer learning. We will here elaborate what this mean by using similar notation as in [1] [2]. Given a feature space X of some dimension, usually input data to our model, we will also have a probability distribution $P(X)$ which describe the probable distribution of features in X. A domain d is defined as an input X described by the set of; feature space and a corresponding probability distribution $D_d = \{X, P_d(X)\}$. However, we don't need to know $P_d(X)$ to infer that it is a domain, only that it exist and are different from some other $P(X)$. Now, say we have an output data defined by a label space Y. It will be described by a conditional probability distribution $P(Y|X)$. A task T is then defined as a set of labels given by a specific conditional probability distribution, that is; $T = \{Y, P(Y|X)\}$. In machine learning, if we have a set of examples $\{X_0, ...X_i\}_d$ and their corresponding labels $\{Y_0, ...Y_i\}$, then $P\{Y|X_d\}$ can be approximated by a model $\widehat{P}\{Y|X_d\}$ with supervised learning. This model can be any machine learner, but it will be a CNN in our study. Now, lets say we have a source domain, and a target domain D_s and D_t, with a specific source task T_s and target task T_t. Here we can have four scenarios:

Case 1) $D_s = D_t$ and T_s = T_t.

Case 2) $D_s = D_t$ and $T_s \neq T_t$.

Case 3) $D_s \neq D_t$ and $T_s \neq T_t$.

Case 4) $D_s \neq D_t$ and $T_s = T_t$.

In the first case, the source task and source domain are both the same as in the target. In such case, $P_s\{Y|X_s\} = P_t\{Y|X_t\}$ holds true, so it will become a traditional machine learning problem where D_s will be used for training and D_t will simply become the test set. However, in case 2-4, either the domain or the task is different, such that without a substantial set of training data for both domain most traditional machine learning methods will have a hard time approximating a $\widehat{P_s}\{Y|X_s\}$ equal to both $P_s\{Y|X_s\}$ and $P_t\{Y|X_t\}$. The result is a model biased towards one domain and task, not flexible to do more tasks. Transfer learning is defined as minimizing the dissimilarity between $\widehat{P_s}\{Y|X_s\}$ and the target $P_t\{Y|X_t\}$, when either / or both domain, and task differ (cases 2-4). This can either be done by approximating a model $\widehat{P_s}\{Y|X_s\}$ which tries to fit the target $P_t\{Y|X_t\}$ from the start, or by using $\widehat{P_s}\{Y|X_s\}$ to generate a second model which tries do the same. Transfer Learning is said to be possible if domain and target relates in some way.

Domain Adaptation is defined as the special case of Transfer Learning where case 4) holds true, that is; the task is the same, but the domain is different. An ideal domain adaptation scenario is where $P_s(Y|X) = P_t(Y|X)$ using any X in $\{X_s, X_t\}$, but this rarely holds true, even though the two distributions are related. Thus domain adaptation is relaxed to case 4) and a special case 3) where $D_s \neq D_t$ and $T_s \approx T_t$, but $Y_s = Y_t$, that is; domain is different, task is similar and label space is equivalent.

The reason for $P(Y|X)_s$ and $P(Y|X)_t$ not being the same for any X in $\{X_s, X_t\}$, is due to dataset bias or covariate shift [1]. Covariate shift is when the distribution between two domains has shifted (see figure 2.1 for a demonstration). We say that a model has a dataset bias when it fit poorly towards another dataset since it has been generalized to fit a particular distribution. In other words, it does not generalize well beyond data points outside the scope of its trained domain - there is an unexplored territory (see figure 2.1).

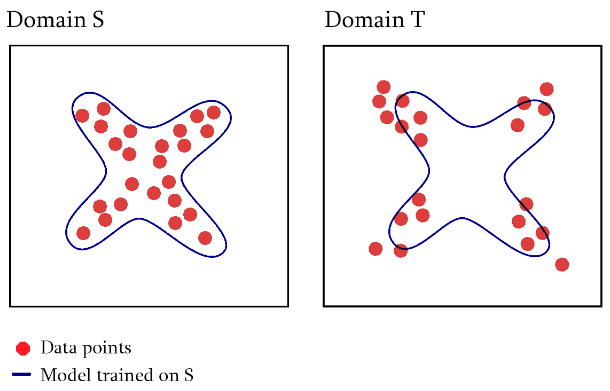


Figure 2.1: An example model trained on source domain S. Because of dataset bias, it does not fit to the target dataset in domain T. We can see that domain T has a covariate shift on its data points, shifting them out and away from the center.

To tie the notations used in this section up to computer vision and CNNs, we can further say that X is a 2D vector (or 3D-4D for video and/or RGB). Each cell is a pixel, and each pixel is considered a feature described by the distribution $P(X)$. This distribution would be different from synthetic data and real data, thus we have two different domains, having the covariate shift influenced by the gap between the real and the virtual. Also, $\widehat{P}(Y|X)$ will be estimated using a CNN.

2.2.1 Overfitting vs dataset bias

Dataset bias is closely linked with *overfitting*, and the term overfitting can be used to describe a strong dataset bias, but we should be somewhat specific. Often in the literature, the term overfitting is used within the context of a singular domain, describing a model which has started to memorize data points - or starts to fit particular nuances within a training set. Here, the training set and the test set is usually from the same domain, and the test set would suffer as it doesn't contain points similar to the training set distribution. Though this is how the term is often used, it can also be used to describe the phenomena when dataset bias becomes so strong that a model loses performance on other datasets outside the training domain. The Oxford dictionary defines overfitting in the following way:

> *Overfitting: "The production of an analysis which corresponds too closely or exactly to a particular set of data, and may, therefore, fail to fit additional data or predict future observations reliably." [3]*

Using this definition, we can see that the term holds true for dataset bias and overfitting given a singular domain. However, since most researchers use the term within the context of a single dataset, unless we specify "overfitting to domain", it can be assumed that overfitting is meant within the context of a singular domain where training dataset and test dataset diverge.

2.2.2 Semi-supervised Domain Adaptation

Domain adaptation is most often only seen as a problem within semi-supervised learning [4]. The reason for this is that given enough labeled data from the target domain, learning becomes supervised, and we should be able to learn a model completely and alone based on samples from the target domain. The amount of data defining this boundary is not clear, as different learners need different amounts of data in order to function. For instance, a CNN needs a huge amount of data to generalize well and don't start overfitting to a domain, often in the thousands, while SVMs can do well in just hundreds of instances.

2.2.3 Few-Shot Domain Adaptation

Few-shot or one-shot domain adaptation is semi-supervised domain adaptation where we only have one or a few labels in target domain [5]. With few-shot learning, as with semi-supervised learning and supervised learning, the exact count of images which distinguishes semi-supervised from few-shot learning is not clearly defined.

2.2.4 Unsupervised Domain Adaptation

In unsupervised domain adaptation, there are no labeled instances in the target domain which can help to reduce the covariate shift between the source and target domain [6]. One simply needs to minimize the gap without getting feedback from the new domain on what concrete steps to take in order to improve. For instance, in object detection, a labeled ground truth example can feed into a learner which positions it got right, reinforcing ”beliefs” which are connected to these positions. However, in the case where no such information is available, the model will not know if it did good or bad, leaving it ”blind”. Unsupervised Domain Adaptation is particularly hard, as learners need to make assumptions about the unseen domain. However, meta-data is usually allowed as ”ground truth”, for instance; information on which data instance belongs to which dataset is allowed. As we will see in the chapter on DA methods, learning features based on ”just knowing that a data example comes from the target distribution” is often the saving factor which can be used to improve unsupervised DA.

2.3 CNN architectures

To understand the results and details concerning our experiment using Mask R-CNN and its general architecture, we will here outline the main ideas behind it. Moreover, Mask-RCNN is based on *Faster-RCNN*[7]. Hence, an overview of the development of R-CNN methods will also be given. Since these are overviews, if the reader is well familiar with the inner workings of R-CNN, Fast R-CNN, and Faster-RCNN, we advise to skip and jump directly to the section on Mask-RCNN (2.3.4). For a description of ResNet, used as our backbone in our Mask-RCNN experiment, we refer to appendix8.3.

2.3.1 R-CNN

In 2014 a team of researchers from EECS Berkeley [8] asked themselves to what extent the well-known architecture of AlexNet [9] relates to object detection. AlexNet was at the time state-of-the-art on the problem of classification, why couldn't the problem of object detection be deduced to a series of classification problems? Here R-CNN was born. It consisted of a smart way to feed in regions of interest, called proposals, into an improved version of AlexNet which would classify it as a background or an object. If you do this on multiple places on the image, you would have a pretty good overview of where the queried objects are. The solution was considerably more accurate than other architectures, achieving a result of 58.5% mAP compared to others with only 30.5% on VOC 2007. The solution relied on Selective Search to generate the proposals, which were fast, but running the improved AlexNet multiple times would be very slow, such that RCNN was not optimized for speed.

2.3.2 Fast R-CNN

As RCNN was showing promising results within object detection, the main issue was related to speed. Since RCNN relied on generating a lot of small sub-images (proposals), which would be fed into the CNN, mainly generated from Selective Search - most of these proposals would be false detections. Thus, a lot of unnecessary computation would be wasted on a large number of bad proposals; resulting in an algorithm spending on average 50 seconds for each image. Furthermore, since a lot of proposals would overlap with each other, similar features would be calculated multiple times. In 2015, Ross Girshick working for Microsoft Research expanded upon the idea that similar proposals could have similar features, thus you would only need to calculate all features once [10]. Here an idea of pre-calculating all features was born, resulting in a new architecture based on the concept of a *feature map*. The feature map would be similar to the features from RCNN, however, the convolutional operations would be executed on the entire image in one go. Here, the output would be a large feature map where each data point in the map would correspond to an area in the input image, such that that local information would be kept. With this ”upside down” architecture, the need for proposals would not decrease, as you still needed to generate an equal amount of proposals on the feature map instead of the image itself, now called ”Region of Interests” (RoI). However, since all convolutional operations have already been calculated, processing each RoI would now only involve some few calculations from ”headers” which would be considerably less expensive. These headers involved some fully connected layers with a softmax classifier and a bounding regressor to improve the accuracy of the bounding boxes for each object. Fast-RCNN outperformed regular RCNN by a factor 25 times faster, with an average of 2 seconds for each image while maintaining the same mean average precision (mAP).

2.3.3 Faster R-CNN

In January of 2016 a paper called *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks* was released from a Microsoft Research team in China [7]. The paper drew some popularity due to its upgraded speed improvement on object detection compared to other state-of-the-art R-CNN approaches. The architecture was based on the same principles governing Fast-RCNN, however, the core improvement was to add a region proposal network (RPN) generate RoIs, but also cutting down on the number of RoIs fed into the top classifier and bounding box headers. Where Fast-RCNN relied on a separate algorithm to generate RoIs, more specifically Selective Search, the team discovered that the number of regions outputted from this algorithm was the biggest bottleneck of Fast-RCNN. At the current time, thousands of RoIs would be fed into the header classifier and bounding box regressor for each image; a bottleneck similar to the bottleneck of that made standard R-CNN slow, here generating thousands of input images instead of thousands of features. Because of this, the team behind Faster-RCNN had the idea that the backbone feature map network in Fast-RCNN could also contain valuable information on telling what parts of the image which could be considered an object, that is regions. The new region proposal algorithm could take advantage of this fact, thus the Region Proposal Network was introduced as another layer. It would work as a gatekeeper to remove regions which would be classified as ”background”. Regions would be generated by using a sliding window of different ratios and sized, called *anchors*, and each anchor would be fed into RPN. As with older approaches to RCNN, using a sliding window to generate different proposals would normally be a bottleneck, at least when each point of interest in the image would generate multiple anchors (e.g as in the original paper, 3 scales and 3 ratios would produce around 2400 anchors). However, the RPN would drastically reduce this number after they were processed. Using features taken from the feature map, each anchor would be processed to get a score on how well the anchor lined up with a real object. Using thresholding on this score one could decrease the number of regions fed into the rest of the network. Since the rest of the network would be considered relatively slow, having a gatekeeper to only select a few regions to pass into the rest of the network would drastically improve the performance. Using RPN, Faster-RCNN would outperform Fast-RCNN on speed on an order of ten.

2.3.4 Mask R-CNN

Since Faster-RCNN and Fast-RCNN all contain a feature map as a backbone, valuable information can be extracted from parts of this sub-network (through RoIs) to solve more problems than just classification and bounding box refinement. This is the main idea through how Mask-RCNN was born [11] (see figure 2.2 for architecture). This architecture is built to solve the problem of instance segmentation. Segmentation is the problem of detecting objects and creating a polygon mask around its silhouette to enclose it. In semantic segmentation, the problem is defined in such a way that it allows overlaps between similar object, but instance segmentation requires a mask around each instance in particular. This is a hard problem, as can be demonstrated by the fact that in 2017 only 5 participants entered the COCO competition on instance segmentation, in contrast to 31 on object detection. Also, in the Cityscapes competition, there were 11 entries on instance segmentation, compared to 58 on semantic segmentation. This gives an idea of how hard the problem is. However, the same year a team from Facebook AI Research tried to solve the problem by extending the Fast/Faster-RCNN architecture with another header. The mask header was added as a third header, parallel to the classification and bounding box, such that all variations of Fast-RCNN and Faster-RCNN could be extended and fit with their architecture (all though Faster-RCNN is the preferred version). The header is a simple, pixel-to-pixel fully convolutional network with 2-4 convolutional layers (depending on what version) and with RoIs extracted from the feature map as input and a float array as output. Each float from the output is evaluated to be between 0 and 1, where the closer it is to 1 the more likely it is considered as a part of the mask. Since the header is working on a low resolution (14 by 14 - 28 by 28), each input-RoI must be stretched to fit the small square input site using a technique called RoIAlign. RoIAlign was invented after discovering that ordinary RoIPool would lose important information from features while they were extracted due to boundaries being extracted at the intersection of pixels. RoIAlign would, however, calculate a more precise RoI, which would not be sensitive to aligning pixels to precise pixels on the feature map. After the aligned RoI would be fed into the FCN header, and a mask is predicted, the output mask needs to be fit back to the original position, with correct size and perspective. This can be done by storing the reverse transformation information from RoIAlign and applying it to back to the output mask.

Mask R-CNN can be trained in one step, that is; using backpropagation end-to-end. However, since the architecture is complicated, involving a lot of dynamic parts and is built by 5 major components (*backbone feature extractor*, *Region Proposal Network*, *class header*, *bounding box header* and *mask header*), the network have some unique behaviours not represented in all neural network architectures.

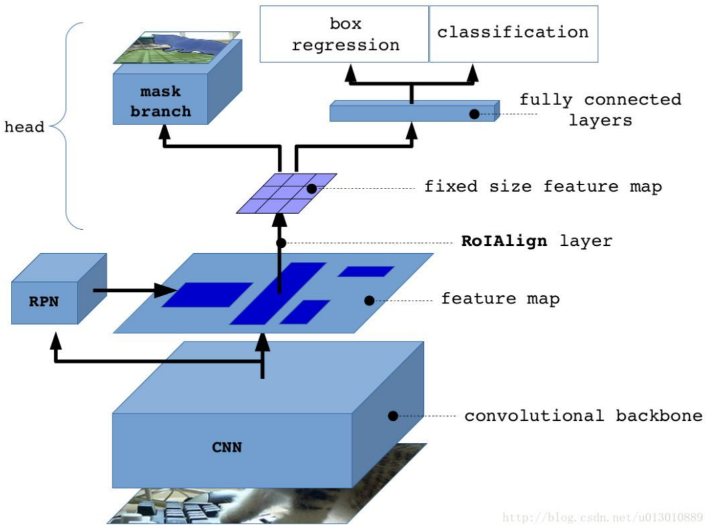


Figure 2.2: Architecture of Mask-RCNN. We can see the feature extractor at the bottom of the image, followed up by RPN set to predict RoIs (seen in dark blue). At the top, we have overheads, which consists of a bounding box, classification and the newly introduced mask header specific to Mask-RCNN. Source Image: [12]

With Mask-RCNN, any fully convolutional neural network can replace the backbone feature extractor, such as GoogLeNet, VGG16, Feature Pyramid Networks, ResNet etc.

Chapter 3

Synthetic Data

In this chapter, we will take a look at synthetic data used in literature, why it is preferable, and how its use affects CNNs. This chapter will work as a build-up for our literature study on domain adaptation, as we will see why domain adaptation is needed, and what exactly we are trying to solve with DA.

3.1 Why synthetic data

3.1.1 Large datasets

Deep neural networks need vast amounts of data to keep local minima from being centered around overfitting solutions or have a strong favor towards in-sample features. To acquire data for CNNs we can mention the physical aspect of taking pictures, filming or harvesting pictures online, but also the post-work of cropping, labeling and producing annotations. This is considered a repetitive, laborious task, and this means that huge datasets needed to train CNNs are hard to come by. An alternative method to such laborious work is using synthetic data which can be is easier to come by in huge quantities due to automatic labeling. We will take a look at such methods in the next paragraph.

3.1.2 Generating synthetic data

Several methods of generating synthetic data have been proposed. In this master thesis, we are interested in synthetic data generated from computer graphics, which represents a new frontier in visual applications. One way of harvesting such data by computer graphics was proposed by a German team from the University of Darmstadt[13]. The team injects a wrapper between a game running on a PC and the operating system, allowing them to record, modify, and reproduce the rendering pipeline. This could turn big open world games such as *Battlefield*, *GTA5* and *The Witcher*, to labeled datasets. With this technique, the team successfully extracted 25 thousand segmented images from the game GTA5. However, because the method was not dealing directly with source-code, one couldn't access labels of objects within the game engine itself. Adding label-tags would still have to be done manually. However, since each segment was associated with an ID, labeling only had to be done once for an entire video sequence. Simply clicking on one example of an object for labeling, would make a script take care of all the other examples (see figure for example of labeled data with corresponding synthetic images3.1). In this approach, all objects in 25 thousand images of an urban environment were perfectly segmented and labeled in just 49 hours, which is 141 milliseconds on each image. According to the researchers, fully manually annotating and segmenting similar detailed urban environments for hand, would take about 90 minutes for each image. This approach proves just how effective a synthetic data generation can be, speeding up the work process by a factor of 40 thousand.

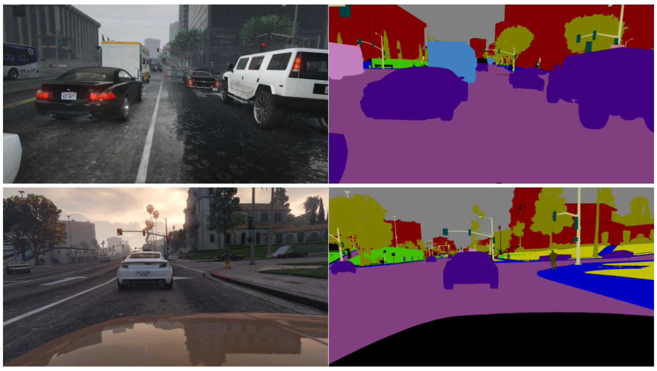


Figure 3.1: Segmentation ground truths with corresponding images from GTA5 using the approach from the University of Darmstadt. Image source: [13]

Academia and industry are picking up on the usefulness of computer graphics as an alternative approach for machine learning. For instance, in autonomous flight, Microsoft newly created AirSim [14]. In the world of robotics and autonomous vehicles, good computer vision algorithms are one of the biggest challenges for secure and predictable behavior. To reduce the risk of destroying expensive equipment in testing, and to speed up training time - robotic simulators represents a way to ease these technical challenges. Robotic simulators such as Gazebo [15] provides excellent physics and robot-world-interaction but is not state-of-the-art in computer graphics for visual applications. The team from Microsoft, looking for a good simulator for autonomous drones, noticed this discrepancy between the great robotic simulators and the lack of good computer graphics to aid computer vision algorithms. They proposed using Unreal Engine 4 - a state of the art game engine. Similar to the approach proposed by the German team for computer games above [13], AirSim comes with an API which can generate segmented images of each unique object in a scene. Each unique object is color-coded with a unique color which can be thought of as an ID number. This ID can be used to identify the class and position on each object. Such features are practical when making labeled data. Furthermore, the graphics card company Nvidia has also recognized synthetic data to be the future of robotics. They have newly launched its early access to *NVIDIA Isaac*, a robotic simulator also based on Unreal Engine - designed to also include computer graphics as an alternative approach for training visual machine learners [16].

Simulation environments as those described above, reduce the line between the real and the virtual for AI systems. However, computer graphics do not need to be associated with an entire virtual world to aid machine learning. For instance, a particularly creative approach was done by a team from the Technical University Kaiserslautern in Germany[17]. The team proposed using 3D-scanned objects or downloaded CAD-models which they wanted to use for training in object classification. Here they made a software capable of generated thousands of images from all angles of the CAD-models, with random backgrounds for each image (see figure 3.2). The reason behind this approach was for capturing maximum information as possible on each model to be used in a CNN.

An additional creative approach came from the Israel Institute of Technology department of computer science [18]. Set on a task to learn 3D-reconstructions of faces based on single images using a CNN, they needed a huge amount of 3D-annotated images of faces as training data. To produce such images, they used a 3D Morphable Model, a statistical model which can generalize many faces in different expressions, ages, gender, and ethnicity. The model was able to produce an infinite amount of faces with corresponding ground truth-data. Their CNN performed very well in the real world, by just training on synthetic data.

In conclusion to the task of acquiring training data, from all these use-cases mentioned, it is clear that with computer graphics, labeling is easy, and the desired use-case is almost limitless, only restricted by the will and skill to model a 3D-environment. However, the question still remains how CNNs, in particular, are affected by such data. This will be discussed in the next section.

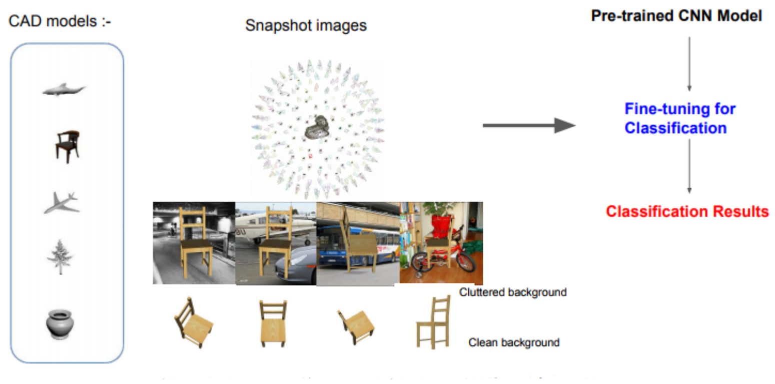


Figure 3.2: Synthetic data generator approach, generating images of 3D-object from many angles with random backgrounds. Image source: [17]

3.2 Synthetic data and CNNs

3.2.1 Some inspiring use cases

We will here see that synthetic data generated from computer graphics can indeed be used to aid CNNs. For instance, in 2016 a team from the University of Princeton University used deep learning in a semantic scene completion task [19]. The challenge was to convert colored depth-images into a 3D-model of the environment using voxels (small unit blocks), where each object in the environment would be tagged with a class label. The team designed their own CNN, which was unique in how the last layers used *dilatation-operations*. These dilation-operations would help the CNN to gather contextual information of surroundings to get a deeper understanding of classifications. To train this CNN, a huge database of apartment depth-images was needed. Being short on such data, the team decided to manufacture it by themselves.

> *Our SUNCG dataset contains 45,622 different scenes with realistic room and furniture layouts that are manually created through the Planner5D platform. Planner5D is an online interior design interface that allows users to create multi-floor room layouts, add furniture from an object library, and arrange them in the rooms*

A generator producing depth-images from partially, random, camera angles, virtually placed in these room layouts, would feed 130,269 depth-images into the CNN. Compared to the same CNN trained on real data, there was a set back of 2-3% on both precision, recall, and IoU. However, this setback is remarkably small considering such model performed "in-the-wild" completely trained on synthetic data. The team also tried to mix both synthetic and real data. Here, they trained one week on the synthetic images, and 30 hours on a small set of real images. Combining both datasets, beat both previous techniques by 2-3% in precision and recall, and gave a 10.3% increase in IoU.

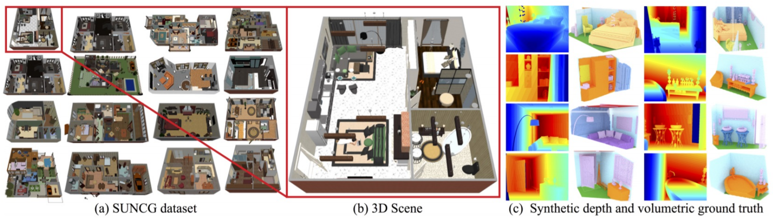


Figure 3.3: SUNCG dataset. Synthetic data generated from virtual apartments. Image source: [19]

To further add to this research. In fall of 2017, a paper came out from the Indian Institute of Technology, where a small team of scientists was interested in how synthetic data affects CNNs [20]. Their task was to perform object detection on food and drink in a refrigerator only using synthetic data. The team used GoogLeNet architecture, pre-trained on weights from the ImageNet database which was later, again, trained using 4000 synthetic images generated with Blender, a 3D-software known to produce realistic computer graphics. Here, every food, drink - and refrigerator 3D-model was downloaded from the internet. In their research, they discovered that the CNN trained on synthetic model performed with 24 mAP when it was tested on ”in-the-wild” real images. This was slightly under-performing against the same architecture when it was trained using only 400 real images, which achieved 28 mAP. However, this was quite good considering no real data was used and there was only a relative loss of 14%. There could also be a larger gain if the team used only 2000 synthetic images, as their findings suggested too much synthetic data can actually overfit to the synthetic domain. Also, the team wanted to investigate how mixing synthetic training data with real training data would affect the performance. Mixing the two databases with 3600 synthetics and 400 real (not from the test set), beat both models by 12%, achieving an impressive 36 mAP.

3.2.2 The synthetic covariate shift

When testing CNN models trained on synthetic data on real-world images, the results from the papers investigated in the previous section showed a quite small relative setback (only 2-3% and 14%). Such a small setback shows that the covariate shift between computer graphics and real images can be quite small. However, this is not always the case. When leading a small research team from Nvidia in July 2018, Dundar [21] tested various domain adaptation techniques on synthetic data. The team showed that a gap from the synthetic to real can be much broader than in the cases in section 3.2.1. In the context of a semantic segmentation problem - going from the GTA in section 3.1.2), to a similar real-world driving scenario dataset called *Cityscapes*, the performance dropped from 66.3 IoU to 22.9 IoU compared to a CNN only trained on images from the real world. The CNN architecture used was *Dilated Residual Network DRN-C-26* which were based on ResNet and dilated convolutions. Testing on a different CNN (DeepLabV3) with another synthetic dataset called Synthia as the training set, the new model only achieved 18.5 IoU. The team also tried using the SUNCG dataset (mentioned in 3.2.1) as a synthetic training dataset for indoor scenes (using it for semantic segmentation instead of 3D-scene reconstruction). Applying the trained CNN to a real-world indoor domain, the model dropped from 50.4 IoU to 17.9 IoU, compared to a model trained only on real-world indoor images. These findings can further be supported by results from our 2017 project thesis [22], where a synthetic dataset, modeled completely to fit position, shape of objects, camera angles, and light conditions from a reference real-world dataset, still resulted in a drop from 82 mAP(%) to 29 mAP (%) compared to the model trained only on real-world images (see figure 3.4 below for demonstration image and details). These experiments show that even though computer graphics can be state-of-the-art in terms of realism, and look very similar to the real world, there is still something which can hinder the full potential of it when deploying CNNs trained on synthetic data to the wild. This can possibly be attributed to features unique to a computer graphical environment, which is not present in the real world. Even though humans have a hard time differentiating between a fake and a real image, it may be that a CNN trained on synthetic images ”sees” different than us and can pick up on these unique features. On this topic, some possible unique features will be discussed in our experiment in section 5.2.3. Wrapping up our findings, we can say that a covariate shift from the synthetic to the real world is probable, and this represents a big problem when relying on synthetic data for real-world cases. Even though a model can possibly perform good in-the-wild (as we have some examples of referring to section 3.2.1), there is a large risk that it can't. Unfortunately, this means that synthetic data can't be fully potentialized before the covariate shift is reduced. This brings us to the field of domain adaptation, where many methods and techniques have been proposed to aid us in such problems.

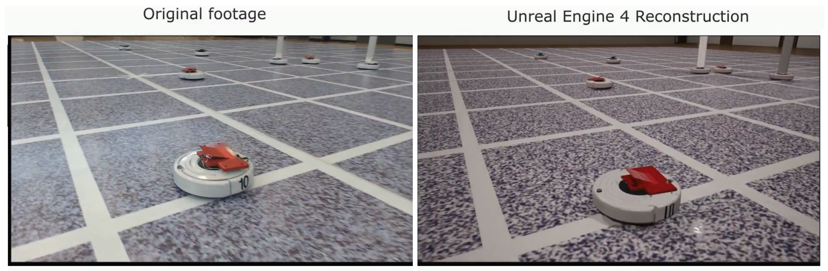


Figure 3.4: A dataset containing 1089 images was made in Unreal Engine 4 to perfectly mirror a real-world dataset containing Roomba vacuum cleaning robots inside a gym arena. Even though the two datasets were completely similar, a CNN (YOLO [23]) trained on the synthetic domain to detect Roombas, did not perform well in the real-world domain, showing computer graphics contain features building up to a dataset bias. However, some overfitting were present in the experiment.

Chapter 4

Literature survey: Domain Adaptation

4.1 How domain adaptation relates to synthetic data

Given that computer graphics is useful, but substantially different from real-world images (referring to section 3.2.2) - at least enough to negatively affect performance when switching to a real-world test set - methods and techniques are needed to combat this effect. When deploying a CNN trained on synthetic images on a dataset consisting of real images, the following cases holds true; $D_s \neq D_t$ and $T_s \approx T_t$, but $Y_s = Y_t$, thus the problem can be reduced to a domain adaptation problem (see basic theory section 2.2). In general domain adaptation, many methods and techniques have been proposed to reduce the covariate shift and line up models to fit the target domain - or line up both domains. However, not all DA methods have been tested with computer graphics, instead, most are using the Office dataset [24] and switching between MNIST [25], SVHN, and USPS for testing DA methods and techniques. The former dataset consists of pictures of products seen from a web-store domain in contrast to products seen in a house setting, while the latter datasets are handwritten numbers in contrast to street-sign numbers. Though these are the more normal datasets used in literature, since synthetic data by computer graphics vs real data can be defined as a DA problem, it is reasonable to think that the same methods and techniques would be applicable to computer graphics. To answer our thesis problem definition, we will in this chapter take a look at various methods and techniques for domain adaptation. Here, techniques which are not applicable to CNNs for computer vision applications will be filtered out. We will start by presenting fine-tuning and off-the-shelf methods and dive into how CNNs behave using these methods. Further, we will look at advanced DA methods used in CNNs. Lastly, we will discuss how DA can be used for most CNN visual applications. It is important to mention that the chapter is discussing general DA techniques, such that it will be useful in most scenarios, and not just synthetic data. Though some of the papers presented here have tried DA with computer graphics in relation to real data [26], a comprehensive experimental study on how all DA techniques compete against each other using computer graphics, is not present to our knowledge. Experimental support would be preferable if such can be identified.

4.2 Classical methods

4.2.1 Fine-tuning

Fine-tuning is considered a universal method for all end-to-end neural networks. This is because fine-tuning is simply introducing more data for training on the same model we started with. This data is usually taken from a new dataset which would be considered the target domain in DA, or real-world data (for the synthetic-to-real DA scenario). Fine-tuning is often accompanied by the technique of freezing layers where we want to keep weights from updating, usually, because we don't need these layers to fit the new target domain as they are "general" (more on this in section 4.3.1). It is also possible to reset some layers back to random, and fine-tune the whole network. Furthermore, fine-tuning accompanying it with data augmentation is recommended if data is scarce.

4.2.2 Off-the-shelf methods

Off-the-shelf methods are a very commonly used technique for transferring knowledge between CNNs and is used in both transfer learning and domain adaptation [27]. The name of this technique is a metaphor for looking at the CNN as a "shelf", then we "take one shelf off" to use the remaining architecture for our target task. In all essence, it is reusing weights in a pre-trained network which is considered to be "general", but taking away weights and parts of the network which is domain or task specific. In a CNN, this is usually the deepest layers which we will discover in the next section 4.3.1. This means that the pre-trained sub-network will act as a feature extractor, where each feature can be used as an input in a machine learner which tries to learn the new task. This machine learner is not limited to neural networks, as SVMs, decision trees etc. can be used. However, it is very common to use a neural network initialized to scratch with random weights. Thus, the whole architecture set for transfer learning becomes a normal CNN with random weights at the deepest layers and frozen layers in the shallower parts. In domain adaptation, it would be natural to use the same architecture on target domain as in source domain, thus in DA off-the-shelf is similar to fine-tuning. In DA, off-the-shelf use random weights on the deepest layers and keep pre-trained nodes frozen, while fine-tuning can describe all cases where we keep many weights from the source network [28]. Off-the-shelf is considered to be a classical approach to transfer learning and domain adaptation, as it is an intuitive common technique and has been around for a while. It represents a fast, and easy way for domain adaptation, and performs quite well [27].

4.3 Fine-tuning and transfer learning of CNN

4.3.1 How transferable are features

To get a more intuitive understanding of how neural networks behave under domain adaptation, we will here take a look at a paper which was published in 2014 named "How transferable are features in deep neural networks". It was a collaboration between multiple American universities and led by Yosinski [28]. Here, researchers used Off-the-shelf and fine-tuning to investigate the behavior of a CNN during transfer-learning, but isolating parts of the network to understand the anatomy. Such knowledge becomes important when we want to discuss and predict the behavior of our CNN during fine-tuning in our experiment in chapter 5, but it also works as a basis for the fundamental anatomy of a CNN, used in all DA / transfer learning methods. The paper shows that domain adaptation becomes harder when we want to transfer high-level features located in the deepest layers, while earlier layers are general and can be used for most DA scenarios. The experiment put forward in the paper involve transfer learning, but as discussed in section 2.2, DA and transfer learning are similar in nature. In the paper we can read the objectives:

- *Can we quantify the degree to which a particular layer is general or specific?*
- *Does the transition occur suddenly at a single layer, or is it spread out over several layers?*
- *Where does this transition take place: near the first, middle, or last layer of the network?*

To answer the questions listed above, Yosinski attempted transfer learning on the dataset ImageNet from 2012 - a classification task with 1000 different classes, containing 1.3 million images. The dataset was split into two, where one dataset consisted of images depicting 500 different types of classes, named dataset A, while the other dataset contained the rest of the 500 classes, named dataset B. Here they did supervised transfer learning from A to B, that is; learn to predict classes in B - based on features/weights discovered by a CNN trained on A. Using an AlexNet trained on A, here called *base-A*, the team used off-the-shelf to reset layers above a certain "shelf" to random, then *relearn* these layers based on features fed into them as a product of images in dataset B. The position of the "split", in terms of depth, was consecutively tried from layer 1 to layer 7, making seven different transfer learning models $A1B, A2B...A7B$. A second experiment was attempted, where all conditions were similar, but no freezing was used, making all layers open for fine-tuning. This resulted in an additional transfer learning models $A1B^{+}, A3B^{+}...A7B^{+}$. By testing performance on a test-set from B, one could compare and see what layers contained features which were general (common for both A and B), and what layers were biased (only working for A). The results from the experiments can be seen in figure 4.1.

Based on the data in figure 4.1 the team made the observation that the first layers in a CNN are highly general, making them "universal" for both source - and target domain and task. It is well-known that the first layers in CNNs often converge into Gabor Filters and Color blobs during training [28]. Such filters are very useful for many computer vision tasks, so it was expected to see a smooth transition when reusing these features on B.

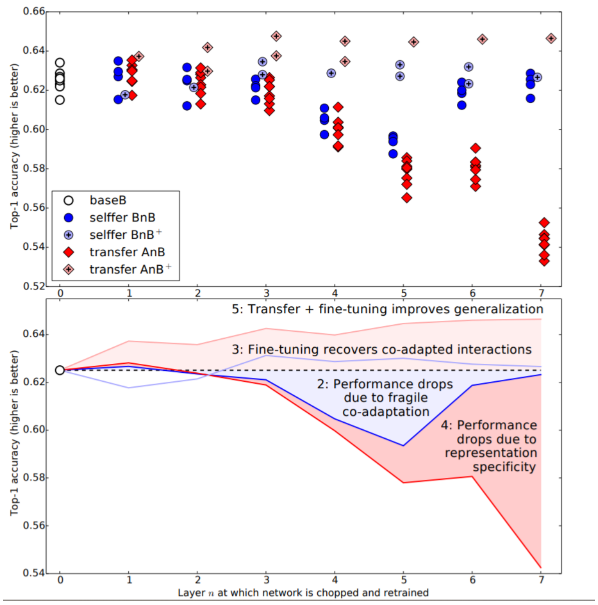


Figure 4.1: Results from the paper "How transferable are features". On the X-axis we can see models $A1B$, $A2B$...$A7B$ and $A1B^+$, $A3B^+$...$A7B^+$ in red, with performance measured on Y-axis. Here, *baseB* is a control model purely trained on dataset B which can reveal if transfer learning is effective. From the red data points without crosses, we can see that regular off-the-shelf, chopping off layer 7 (resetting layer 8 to random and train on B), will underperform compared to chopping of shallower layers. This is because, in supervised transfer learning, there is enough data in B to relearn higher concepts, but freezing weights from *base-A* will force the network to be less adaptable. Keeping layers 1-3 gave good performance on the test set B due to them being highly general across tasks. In the case of "no-freezing" (red crosses), keeping shallow layers for transfer learning resulted in good performance across all "splits" due to them being able to use all layers. A small boost in performance across all these could be attributed to the pre-training on A (some layers were trained on more data). The blue data points, *selffer BnB* and *selffer BnB+* represent procedure control models, doing all experiments only using dataset B as source and target to make sure performance is not due to the training algorithm itself. These control models showed that for off-the-shelf methods, co-adapted interactions between neurons on opposite side of the "split" is important, and can actually be broken because of freezing. Image source: [28]

The data also seems to suggest that the deeper you go into the network, the more layer-activations / features seem to be biased towards pre-trained source domain and task. We can imagine that as we move deeper into a CNN network, task/domain-specific abstractions are accumulated and kept for end-result predictions.

Furthermore, the team discovered that co-adapted interactions between neurons across layers is important and can actually be task/domain specific (see figure 4.1 text for an elaboration of how this was discovered). This means that resetting some layers to random can make surrounding layers underperform. Here, we can expect that source task performance linger upon such connections being intact, making freezing and training on a new dataset somewhat risky, as it can remove task/domain-specific co-adaptions between layers, thus underperform on both domains. This can also be seen as not actualizing the full potential of the frozen pre-trained network, as it relies on deeper layers.

As for what DA technique is recommended in general, the experiment couldn't tell us what off-the-shelf technique was best in terms of semi-supervised - or few-shot transfer learning. The reason for this is that this experiment was only done with a large target dataset. However, this was further investigated by another team, presented in the next section.

4.3.2 Fine-tuning and freezing

In 2016 a team from the University of California, Berkeley tried to study the behavior of fine-tuning in DA scenarios (a.k.a. Chun[29]). They investigated the best practices for fine-tuning with or without freezing, and/or with or without random initialization. At the same time, they varied the training set size of the target domain dataset, and the type of target domain, where each target domain was gradually more different from the source. With 138 experiments, mapping out these relations, the results seemed to suggest, that irrespective of dataset size and how different the domains were, keeping most layers is preferable. This means that randomly initializing any layer below the last layer is almost always inferior to the alternative. The team did not compare "off-the-shelf with the last layer", with "fully fine-tuning", so results must be seen in relation to this fact.

Taking only the top layer off (initializing it to random), the team went further and investigated if fine-tuning only on the top layer (freezing layers below the top layer), or if fine-tuning all layers was best. They found that the more target data used, the more freezing would be inferior. Also, they found that as the target domain was more and more dissimilar to the source domain, the more freezing was inferior. This means, in a few-shot domain adaptation problem where the top layer is initialized to random, freezing is recommended, but if the target domain is very dissimilar then all layers should be fine-tuned.

4.3.3 Style-transfer (adversarial reconstruction)

target domain data is scarce, data augmentation can be considered as a way to synthesize more data, increasing the scope of classical methods in semi-supervised and few-shot DA. Additionally, new methods in deep learning allow for more advanced ways of synthesizing data, now even without labeled target domain examples as a prerequisite (unsupervised DA). The development of Generative Adversarial Networks (GANs) has allowed for some interesting use-cases within computer vision. Such networks are known for their power to generate realistic synthetic images. They can use noise as input to generate an image, but also perform image-to-image translation [30]. In 2015 a paper was published called *A Neural Algorithm of Artistic Style*. Here Gatys [31] introduced a neural network which could take in an image as input, and output the same image modified to look like a painting with the artistic unique style of a famous painter such as Van Gogh, Edvard Munch etc. This launched a new branch of research which tried to optimize style-transfer between domains using neural networks, but not just painting-styles, also others; such as day to night conversation, horse to zebra conversation etc. Here, GANs are especially good at finding such mappings. One such GAN is CycleGAN [30] which is state-of-the-art for discovering high-level feature mappings between images. However, using CycleGAN some training is required to learn new styles. In 2017, *Deep Photo Style Transfer* [32] put forward by Adobe showed that style transfers can be done using a single input image and a single style-reference photo (see figure 4.2).

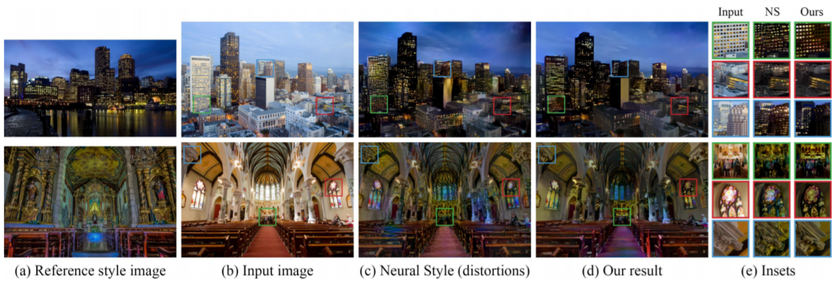


Figure 4.2: Example images from Adobe's style-transfer technique. Using a reference style image and a input image, color and contrast can be transferred from one image to the other, and still look realistic. Source: Adobe [32]

The same year, Li [33] theoretically showed that the problem of finding mappings between an input image and the new style in style-transfer corresponds to techniques used in domain adaptation. If such mappings can bring two domains closer, could data augmentation reduce the covariate shift between source - and target training data? At the beginning of 2017, Csurka [34] tried just that. Using styles from the target domain, the team behind Csurka generated a dataset containing source images with labels from the source dataset, but style-transferred to look like the target dataset. The idea was to ”fool” the CNN to ”think” it was looking at labeled images from the target domain. If it could be demonstrated that style-transferred source images are considered ”similar” to target images in a CNN, the natural consequence is that labeled data in the target domain can be easily generated, and plentiful. Surprisingly, domain adaptation using such data worked quite well. Csurka compared such method with discrepancy methods for DA (we will get to discrepancy in section 4.5). They showed that simply training on the source dataset, then fine-tuning on the style-transferred source dataset, outperformed some unsupervised discrepancy methods by 1%. A similar approach to Csurka was put forward by Google under Bousmalis [35]. The team from Google made a GAN which could take in labeled source images and target images, and reconstruct target images with labels similar to the source. This worked well on images with Z-depth labels as ground truth, but also DA on the handwritten number dataset MNIST to the dataset USPS. When compared to the most popular state-of-the-art DA methods of 2017, training on images from Googles GAN outperformed all others on MNIST to USPS DA.

Though style-transfer methods are considered an adversarial method where we reconstruct target domain, the approach for implementing DA into a CNN involve training with such data from scratch, or fine-tuning. Thus, the approach is categorized as a sophisticated classical approach in this study.

4.4 Adversarial methods

Classical approaches to DA can be helpful, but most DA approaches proposed in the literature involves some sort of external modification to a CNN. This does not mean they necessarily need to overwrite the architecture, but it means additionally machine learning parts is added to force such networks to perform well in a target domain. From this point forward in this literature study, more advanced methods to DA will be presented. We will start with adversarial methods which are inspired by how GANs are trained.

4.4.1 Domain-Adversarial Neural Network

A particularly interesting and smart solution for *unsupervised domain adaptation* is the Domain-Adversarial Neural Network (DANN) put forward by Ganin and Lempitsky in a paper from 2015 [36]. This solution to unsupervised domain adaptation can probably be used on all classification CNN architectures using deep feature extractors, and something particularly good about it - without changing the internal architecture. This is possible by adding an external header to the feature extractor, which can - by simple backpropagation into the feature extractor align the distributions of features across the two domains (see figure 4.3 for architecture). The idea is to turn the whole CNN into something similar to a GAN, where the whole network becomes a discriminator and a "feature"-generator. In a way, we can say "the CNN is competing against itself", but how is this possible, and how is this helping domain adaptation? The theory proposed by Ganin and Lempitsky, is that if we force a feature extractor which is set to perform some task T_A, to also learn task T_B; classifying the origin of the domain based on input data, such that $T_B = \{Y, \{P(D_s|X), P(D_t|X)\}\}$ where Y is labeled "source" or "target", then the additional learned task can help the network to learn task T_A better. This is reasonable to think, since knowing such information could help the feature extractor to find a pattern which separates the two domains, and then bring them closer to one another by aligning the features. In practice, this is what it does, and this extra task T_B is executed by an extra "classification"-overhead connected to the feature extractor. In this way, the feature extractor helps both tasks, but in order to do so, it needs to multitask by "aligning domains". Let us call this "aligning" for a sub-task T_S since it is not visible by any output. We can say that task T_B helps the feature extractor to do learn task T_S (identifying the differences between the two domains and align them), but T_A force the same network to still learn the main task. Now, the question is still how one can implement an architecture that will force the feature extractor to learn sub-task T_S based simply on classifying the origin of the domain (task T_B)? Here, adversarial training comes in, forcing the network to compete against itself in a zero-sum game. If the domain-classifying head learning T_B would be called the "discriminator", the feature extractor would be called the "generator", here trying to generate features which makes the two domains seem as similar as possible, working against the discriminating domain-classifier. The main principle behind this approach is that since the feature extractor is trained to make discrimination between domains harder (T_S work against T_B), features that are unique for each domain will lose their uniqueness, thus aligning both domains to produce similar features which can be used in the main task of the network. We can say that the feature extractor is forced to learns domain invariant features, thus performing well on both domains. This can be achieved by multiplying the error backpropagating into the feature extractor from the domain-classifier (discriminator) by minus one. In practice, this reversal operation would become an extra layer named the ReverseGrad (Gradient Reversal layer). It can be thought of as a "pseudo function" defined as:

$$R_\lambda (x) = x \tag{4.1}$$

$$\frac{dR_\lambda (x)}{dx} = -\lambda I \tag{4.2}$$

where 4.1 is the behavior of ReverseGrad during forwarding pass, 4.2 is what happens to partial derivatives that are down streamed through backpropagation, I is the identity matrix and λ describes the amount task T_A is weighted against task T_B during training. Normally, backpropagation would force prior layers to minimize their contributing factor to the total loss of the neural network, helping top layers to perform some task, however in the case of a backpropagated chain executed by ReverseGrad, all prior layers will *maximize* the contributing error, thus making these layers to work against descendent layers. In this way, the feature extractor will start competing against the domain-classifier, even though they are in the same network. Ganin and Lempitsky tested their approach on the standard Office dataset, and outperformed other DA-methods, setting a new state-of-art method of 2015.

Note, even though the architecture is called unsupervised, it only works if input data is labeled with its source and domain (which is true for almost all cases), however for learning task T_A it would be considered unsupervised. Note, the paper does not state if training their model with the discriminator header must be done from scratch, or if it can be added later to a model with weights already trained on some task T_A. However, this is possible with the next adversarial method we will have a look at.

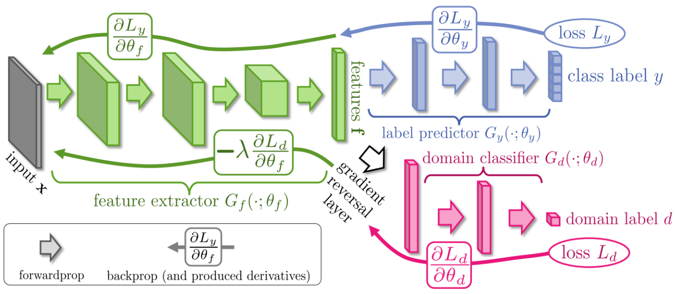

$\frac{\partial L_y}{\partial \theta_f}$
$\frac{\partial L_y}{\partial \theta_y}$
loss L_y
class label y
features f
label predictor $G_y(\cdot;\theta_y)$
$-\lambda \frac{\partial L_d}{\partial \theta_f}$
domain classifier $G_d(\cdot;\theta_d)$
input x
gradient reversal layer
feature extractor $G_f(\cdot;\theta_f)$
domain label d
loss L_d
$\frac{\partial L_d}{\partial \theta_d}$
forwardprop
backprop (and produced derivatives)

Figure 4.3: General idea behind Ganin and Lempitskys Unsupervised Domain Adaptation by Backpropagation. Task T_A can be seen in blue while task T_B in pink. Green is the feature extractor which multitask on minimizing loss from task T_A and maximize loss from task T_B. Source Image: [36]

4.4.2 Adversarial Discriminative Domain Adaptation

A similar principle to Ganins DANN - using adversarial training was proposed by Tzeng [37]. It is known as ADDA, short for *Adversarial Discriminative Domain Adaptation*. It is different to DANN in how the domain alignment is not induced in one single network, as it would in DANN, but instead between two separate networks, one source-network, and one target-network. This means that there is no shared training pipeline in the source and target feature extractor (see figure 4.4 for architecture), training is instead divided into three stages. Here, the target-network's feature extractor, or here called *encoder*, is set to learn a mapping from target domain images to source domain features, or in other words; to mimic source features similar to features outputted from the source-network-encoder. The model is not trying to learn domain invariant features, rather focusing on making the target-encoder optimized for target domain images only. The three-stage training approach is similar to how GANs are trained, and particularly GANs set to synthesize realistic images. To understand ADDA, a brief overview of such training will here be given:

When training a GANs, a realistic image is presented to a discriminator and would be classified as real or fake. If it gives the wrong prediction, the loss will backpropagate into the discriminator, improving it over time. If it gives the correct prediction, it will reinforce weights contributing to this. A generator neural network will try to present a fake image to the discriminator. Here, the discriminator will spot some features which will help it to judge if the image is real or fake, improving it over time. However, the loss is also propagated into the generator to improve its parameters, either by reinforcing weights that contributed to the discriminator loss or penalizing weights that helped. Over time, the generator will become better and better to fool the discriminator.

As proposed by Tzeng, this principle can also be used in domain adaptation. Instead of a generator, we have an encoder, and instead of a discriminator which classifies fake or real images, we have a discriminator which classifies if the features come from the source-network or target-network. Training is divided into three stages:

1. Train source-network to learn normal classification in a supervised manner on source images as we would in normal classification tasks.

2. Copy weight from source-encoder with its parameters to the target encoder for initialization of the target-network. The target encoder will from this point only receive target images as input. Now, chop off the source-network overhead, only keeping source encoder, then feed source encoder and target encoder into a domain discriminator. Use adversarial training on the whole encoder-discriminator system, and validate the target encoder on its ability to impersonate the source encoder. Only backpropagate into the target encoder, as source encoder is fixed.

3. When the target encoder starts presenting to the discriminator features only similar to source features, it has finished training. Then, use the target encoder as input to source classifier.

In (2), since backpropagation only happens in the target encoder, it will be forced to transform internal weight to map to features similar to source features output. The discriminator will eventually have a hard time to separate the two. In (3), Tzeng proposed that if the discriminator has a hard time separating features, then its natural to assume the source classifier overhead will have the same problem, thus the source classifier overhead can be used with the target encoder.

Testing this approach on the MNIST dataset, going from MNIST to the USPS dataset, and the reverse, the approach did 14.7% better than DANN on average.

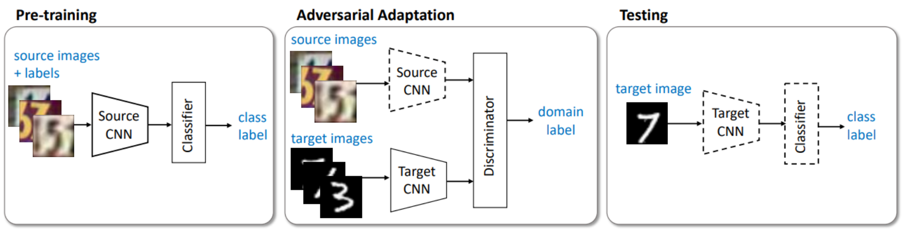


Figure 4.4: ”An overview of Adversarial Discriminative Domain Adaptation (ADDA) approach. First, we pre-train a source-network CNN using labeled source image examples. Next, we perform adversarial adaptation by learning the target encoder to imitate source features, such that a discriminator cannot reliably predict the origin of each domain. During deployment of the model, target images are sent into the target encoder as input, they are mapped to look like source features and then classified by the source classifier. Dashed lines indicate fixed network parameters (freezing).” Image and caption text source: [37]

4.5 Discrepancy based methods

Adversarial methods for DA are elegant in how they use backpropagation to force networks to learn features similar to both domains. In discrepancy DA methods, however, the CNN is instead constantly monitored to check how its inner workings behave differently across domains using a *discrepancy measure*.

4.5.1 Maximum mean discrepancy

The Maximum Mean Discrepancy (MMD) is a kernel-based statistical tool which can be used to calculate the distance between two sets of probability distributions [38]. Since a true underlying distribution is often not known, it can be approximated numerically with the MMD. The MMD has been widely popularized as a tool for machine learning in as it can define a difference between two sets of data points as a singular absolute value. This brings us into the field of domain adaptation. Here, it was thought that if we can measure the discrepancy between the two domains, then this value can help us overcome domain discrepancies by minimizing the difference between the two domains. It has been used to both select - and weigh training examples selected from the source domain based on similarity to target domain [4], and it has also been used as regularizer which can help to minimize domain discrepancy during training. In the context of domain adaptation, the MMD is defined in the following way:

$$MMD(X_S, X_T) = \left\| \frac{1}{X_S} \sum_{x_s \in X_s} \phi\left(x_s\right) - \frac{1}{X_T} \sum_{x_t \in X_T} \phi\left(x_t\right) \right\| \tag{4.3}$$

where x_s are data points in the source domain, x_t are data points from the target domain and $\phi(x)$ is a transformation in Reproducing Kernel Hilbert Space (RKHS) [26]. The transformation $\phi(x)$ can be chosen freely between a set of functions as long as they are in RKHS, that is; all pointwise evaluation of $\phi(x)$ must be continuous linear functional. Furthermore, the two summation terms divided on dataset size in the equation 4.3 is responsible to generate a mean out of all data points. Here, subtracting the target mean from source mean gives us a distance for each feature. The last operation, the norm, gives us an absolute value in distance, thus we have a measure of discrepancy between all data points. Be in mind that X_T and X_T in the MMD does not need to be input data (e.g images). MMD can also be applied to data points which is only depended upon X_S and X_T. In other words, this could as well as be done to abstract representations of data, such as hidden layers in neural networks, as well as it could be done to the first layer. Naturally, this means that MMD will come in useful for CNNs. In the following section, we will take a look at some approaches to deep domain adaptation where MMD has been used, but also where statistical measures similar to MMD has been used.

4.5.2 Deep Domain Confusion with adaptation layer

A paper often referenced in the literature is Tzeng's, *Deep Domain Confusion: Maximizing for Domain Invariance* [38] (DDC), a semi-supervised and unsupervised architecture for domain adaptation put forward in cooperation between Berkeley University of California and UMass Lowell. The team proposed a Siamese network with shared weights which uses the MMD as a regularizer to learn the differences between domains (see figure 4.5. A Siamese network is basically two neural networks sharing some weights by referring to the same address in computer memory when accessing these weights. Though the two networks can use different input and be trained differently, they still need to maintain some similar weights. In DDC, the two conjoined networks would be trained on different domains, the first network on the source (but also labeled target instances if available), and the second only on the target. The first network would use standard backpropagation into an AlexNet architecture for solving some given classification task. The second network would try to optimize an AlexNet for maximum domain confusion, that is; the network would become more and more confused to as what domain was fed in, an idea similar to how adversarial DA methods work. Yet, even with confusion at hand, since the architecture use shared weights, the first network would make sure the weights would also contribute to solving the classification task - even when deeply confused. We can say that the network would optimize the following loss function:

$$L = LC(X_L, y) + \lambda MMD^2(X_S, X_T) \tag{4.4}$$

where $LC(X_L, y)$ is the normal classification loss for any given labeled data,

$MMD^2(X_S, X_T)$ is the domain loss, and λ would fraction weighting the importance of domain confusion vs. classification. Furthermore, confusion would be enforced by MMD calculated between features outputted between two none-conjoined *adaptation layers*. These adaptation layers would not have shared weights between the two networks, thus learned features would differ and MMD could not equal to 0. Here, each feature on the adaptation layer would be treated as a data point and contribute to the total MMD, thus a measure for the domain loss could be given and backpropagate into the second network to minimize domain discrepancy. Calculating MMD on whole domains would be computationally expensive, so discrepancy would be executed batch-wise.

As for the layer location and dimension of the adaptation layer; this could be freely chosen, yet the team found that injecting it right before the last fully connected layer in AlexNet would give best results. The team also showed that having the dimension of the adaptation layer small (between 256 and 512 dimensions in AlexNet) could give a ”bottleneck”, which in effect could hinder overfitting.

In 2014, with 81.2% accuracy, Tzeng’s architecture exceeded other publishers methods in the Office Dataset for deep domain adaptation by a large amount, that is; 21

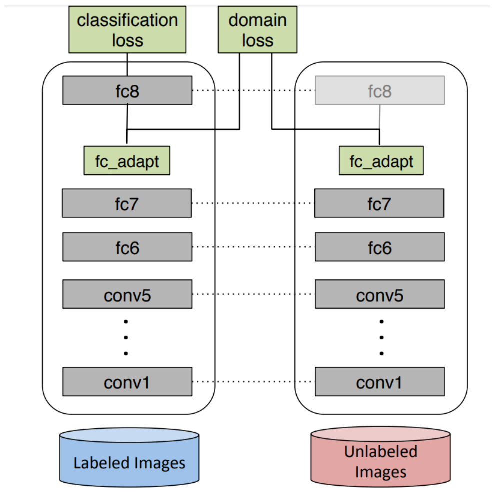


Figure 4.5: Siamese domain confusion network with shared weights. The first network (to the left) takes in labeled images, (generally from the source domain) and the second network (the the right) takes in unlabeled target instances. Fc_adapt is the adaption layer where MMD is calculated between domains. Source Image: [38]

4.5.3 Beyond shared weights

An approach very similar to Tzeng is Rozantsev's *Beyond Sharing Weights* (BSW) [26] was published two years later. The architecture was identical in its general structure, but the team behind BSW suggested some improvements (see figure 4.6 for architecture). Firstly, they felt that having a Siamese network with 100% shared weights between all layers except one (i.e fc_adapt) would force both networks to overly focus on learning invariant features over both domains. Hence, they proposed that some features are specific to individual domains, such that sharing weights between the two network streams, could sometimes force the network to lose valuable, domain-specific features, during training. To contemplate this, the team proposed weights should be related, but not equivalent. Secondly, the team proposed that an adaptation layer would be redundant - MMD should be calculated on all internal layers for a better measure of domain discrepancy. The loss function would then become:

$$L(\theta_S, \theta_T | X_S, Y_S, X_T, Y_T) = L_{class} + \lambda_w L_{MMD} + \lambda_u L_w \tag{4.5}$$

where L_{class} is the classification loss on all labeled instances (domain and target), L_{MMD} is the discrepancy calculated between all layers between the two networks using MMD, and L_w is a weight regularizer which penalizes dissimilarity between weights layer-wise. Furthermore, λ's is hyperparameters; fractions weighting the importance-relationship between the different term regularizes. However, they found that setting all λ_w's to 1 gave the best experimental results.

Considering L_w, even though the team wanted weights to be related, having a L_w loss which mapped one-to-one relationships between all weights in the two networks, could probably make the networks less flexible to subtle changes in input data. Therefore, the team set L_w to penalize the total weight difference between the source and domain *layer wise*. This would allow for a strong relationship between weights for each layer, yet not force every single neuron to act similarly to its twin counterpart. Adding the proposed improvements above, gave Rozantsev [26] a substantial boost over Tzeng on image classification using AlexNet as base architecture. Here, they tested on the *UAV-200* dataset (a computer-generated dataset with images of drones) and outperformed Tzeng by 20% semi-supervised. For unsupervised learning on the Office dataset, they outperformed Tzeng by 2% (0.86% mAP vs 0.84%). Compared to other DA-approaches, Rozantsev always came on top on *UAV-200* and *Office*.

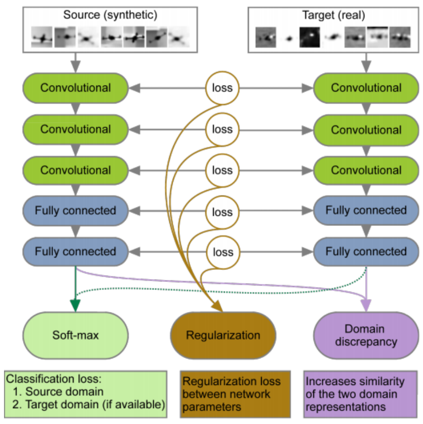


Figure 4.6: Beyond sharing weights architecture. We read: "One stream operates on the source data and the other on the target data. Their weights are not shared. Instead, we introduce loss functions that prevent corresponding weights from being too different from each other. " Image and text source: [26]

4.5.4 Deep CORAL

Deep Coral is similar to DDC and BSW, in how they are based upon Siamese networks sharing weights and deep domain confusion is maximized by a discrepancy measure. However, instead of MMD, Deep Coral [39] uses the *CORAL* loss for measuring discrepancy between layers The CORAL loss is defined as the squared Frobenius norm between the second order statistics (covariance) of two datasets, here each dataset being activation from features on a given layer:

$$l_{CORAL} = \frac{1}{4d^2}||C_S - C_T||_F^2 \tag{4.6}$$

where C_S and C_S are covariance matrices and d is the dimension of input vector data and $||.||_F^2$ denotes the squared matrix Frobenius norm. Using CORAL instead of MMD on a similar architecture as DDC, the method gave an improvement from DDC of 1.5% going between domains on Office dataset.

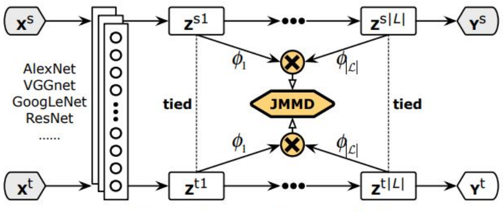
4.5.5 Joint Adaptation Networks

In 2017, a paper came out extending MMD to a new form called *Joint Maximum Mean Discrepancy* (JMMD) [40] and put into a new architecture called the *Joint Adaption Network* (JAN) (see figure 4.7 for architecture). Like all other discrepancy based methods this form of deep domain adaptation used a Siamese network trained on source and target data, with weights related. Here, like Rozantsev [26], Long [40] considered that interactions between multiple layers should be taken into account when doing domain adaptation. However, instead of calculating MMD multiple times, on multiple layers, one could simply take all layers into one single kernel and backpropagate from the loss function into all layers. Before the paper came out, MMD had not been designed for this, thus the team proposed extending it to account for features across more layers. JMMD is defined as:

$$JMMD(P,Q) = ||C_{Z^{s,1:|L|}}(P) - C_{Z^{t,1:|L|}}(Q)||^2_{\bigotimes_{l=1}^{|L|} H^l} \tag{4.7}$$

where P is the network trained on source data, likewise Q is the twin-network trained on target data, $C_{Z^{d,1:|L|}}$ is the joint distribution over all activations from layer 1 to $|L|$ for the CNN.

In general, due to last layers in a neural network being highly domain specific, the team proposed that their idea was superior to previous methods as it would not force these layers to be related on a layer-to-layer approach. Instead, their JAN architecture only made updates to where it was necessary to minimize the JMMD loss. Going back and forth between domains in the Office-31 dataset, compared to DDC [38] the team were on average 6% better on domain adaptation.



(a) Joint Adaptation Network (JAN)

Figure 4.7: In JAN, the Siamese network calculates the JMMD as a function of all layers. Image source: [40]

4.6 Autoencoder Reconstruction methods

4.6.1 Deep Reconstruction-Classification Network

Wang [41] thought of unsupervised domain adaptation for classification of images as a problem which could be solved by a simple autoencoder reconstructing the target domain.

In their paper form 2016 [41], they presented an architecture named DRCN, short for *Deep Reconstruction-Classification Network* (see figure 4.8 for architecture), a network which used an autoencoder to learn invariant features for reducing the domain shift. Autoencoders are networks which try to reconstruct input data. They consist of a neural network with a bottleneck in the middle, forcing the network to compress information into features which then again will be used as input into preceding layers where reconstruction take place. Layers before the bottleneck are known as the encoder, while layers after the bottleneck are called decoder. In visual applications, it is normal to use a CNN as the encoder and a deconvolutional neural network as the decoder. This is also the case for DRCN. Furthermore, for domain adaptation, the idea was to inject a classification learner overhead into the middle of the bottleneck, making the autoencoder into a network of two pipelines; one for solving classification, and one pipeline used for decoding input data - both sharing the same encoder. Their training algorithm consisted of altering between using source data with labels used to train classification header, and target data as input used to force decoder to learn a reconstruction of target input. The method is very similar to DANN in how they both rely on a feature extractor, a classification header, and a separate pipeline to force the feature extractor (encoder) into becoming domain confused. However, in DANN, the domain classifier makes sure the feature extractor is multitasking on learning domain invariant features specifically, while DRCN assumes there is a relationship between features used to reconstruct the target domain, and features used to classify labels for the source. Though this is an assumption, the method worked quite well, even better than DANN by the average of 2.5% when tested on 6 different DA scenarios, suggesting that features used to autoencode target images can have a strong relationship with the source classification task.

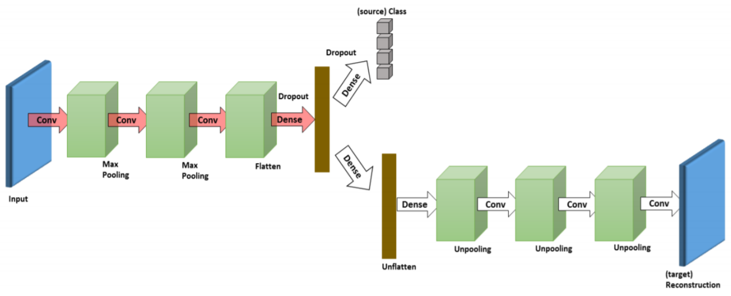


Figure 4.8: Architecture of DRCN. A CNN is used as an encoder for feature extraction. This is shared between two pipelines, one going to the top, and one to the bottom. The top represents a pipeline used to train a classification learner on source data with labels. The bottom pipeline is used to autoencode target images. Making the encoder multitask will bind a strong relationship between the two tasks, thus learning features which can help DA. Image source: [41]

4.6.2 Domain Separation Network

A more advanced autoencoder approach to domain adaptation called Domain Separation Network (DSN) [42] was put forward the same month as DRCN was published - here by Google Research in cooperation with Imperial College London. Its similar to DRCN in principle in how both try to classify source samples and autoencode target at the same time. However, this architecture combines the best of autoencoders with discrepancy methods and/or ReverseGrad. Additionally, the team figured that one single autoencoder was not enough, as it would push features only towards one domain. Instead, they used three; one for the source, one for target and one shared encoder for both. All these encoders would feed their output into a shared decoder. The shared encoder and decoder would be a Siamese network using MMD to learn domain invariant features, but could also be trained to be domain confused by ReverseGrad. This architecture, which is rather complicated, managed to do 12% better than DANN on the SVHN to MNIST DA scenario, and showed that many DA techniques can be combined.

4.7 Survey discussion

We have looked at various methods and techniques for DA in deep learning. Many of which are state-of-the-art. However, most methods and techniques presented in this survey has been designed for classification. This is to be expected, since deep learning architectures has mostly been researched within the scope of this problem [43] [6] [4]. However, some main ideas and principles could still be extended and/ or modified to be used in other tasks. Furthermore, even though many of the ideas in classification were attached to specific architectures it is reasonable to believe their core ideas can be used in other architectures. To elaborate how DA can be used in all computer tasks, we will in this section discuss how these principles can be applied to most computer vision works and architectures. We will go through some common once, classification, object detection, segmentation, and by deduction discuss how DA principles can be extended to these. All propositions laid forward is based on principles we have discovered in this survey, but experimentation should be added to confirm claims. The section on classification will mainly be a summary over many of the DA methods discussed. General DA techniques such as off-the-shelf, fine-tuning and style-transfer will be discussed in their own subsection at the end. It is assumed that these general DA techniques work for all of deep-learning, and will, therefore, be assumed to work for object detection, classification, and segmentation.

4.7.1 The classification problem

The problem of classifying images holds an advantage over many other computer vision tasks. Not just in how it is extensively researched field in deep learning, but also how classification architectures are convenient for DA. For instance, since classification doesn't require an information-rich output, given descriptive features as input, the classification task can be finalized using a single layer or 1-2 dense layers. This means that we can separate these layer from the rest of the network, and focus on designing a feature extractor or encoder which is optimized for DA. In other words, we can make it multitask without disturbing nodes specified for classification. For example, as the principle used in DRCN and DSN (see section 4.6), such designs with a shared encoder can be implemented using one (or multiple encoders) set to multitask - one task is classification and the other is learning features used in reconstructing target domain input. Here, features used to encode target domain images will push weights towards the target domain, then the source classifier is forced to consider the target domain, thus source dataset bias will be reduced. Additionally, instead of using an autoencoder for multitasking a similar principle can be achieved using adversarial learning. Here instead of the target decoder, we have a ReverseGrad layer fed into a domain classifier such as in DANN (see section 4.4.1). This will force the decoder or feature extractor to become domain confused and learn domain invariant features. Moreover, similarly to DANN, one could use generative adversarial training as in ADDA (see section 4.4.2), where instead of learning domain invariant features, weights are mapped towards emulating target domain features. This is done by modifying an existing encoder originally trained on source classification. The modification is achieved by a discriminator set to discriminate between source and target based on features, backpropagating its loss into the decoder. Another approach to DA which is great in classification architectures is discrepancy methods (see section 4.5. The reason for this is that in classification, one can expect information from the whole image being used to produce one single, small, output vector. If all - or mostly all information is relevant, then activation from some, or all layers can be used as input to the discrepancy measure such as MMD (see section 4.5.1) or CORAL (see section 4.5.4). These can measure how similar activation patterns between to forward passes are. Across large batches, such activation patterns can tell us the mean difference between domains. Hence, we can learn domain invariant features by using the discrepancy measure as a regularizer and minimizing it across two streams in a two-streamed network, one for the source stream, and the other for the target. Discrepancy measure can be calculated in one single layer as in DDC (see section 4.5.2), layer by layer using most layers as in BSW (4.5.3), or taking all layers in one go such as the discrepancy measure JMMD used in JAN (see section 4.5.5). Moreover, weights in the twin-network can be completely shared as in DDC, but also have a strong relationship as in BSW and JAN. Finally, a combination of all these methods is possible. This was demonstrated in the more advanced autoencoder DRCN.

4.7.2 Counting objects, measuring area or distance etc

Though classification is a specific computer vision problem, in deep learning, architectures used for solving classification can be reused for other purposes. Most classification problems follow a CNN pyramid architecture, downsampling towards a small output. Such pyramid architectures can be useful in many other deep learning scenarios. It could be counting objects, measuring area or distance, while other cases could be encoding one-hot vectors based on feature representations or learning simple abstract concepts from a whole image. Modifying a classification architecture to fit such tasks, would normally only involve using a different output layer, such as replacing softmax with another activation functions and adjusting the count of nodes in the last layer. Since most domain adaptation methods are designed with minimal dependencies on the last layer, it is reasonable to think problems which can be solved using a pyramid architecture with a minimal information-rich output can also use DA methods presented in this survey.

4.7.3 Object detection - extending R-CNN

In the case of object detection, R-CNN techniques deduce the task of object detections to a series of classification tasks. Thus, it is reasonable to think that all R-CNN architectures could be extended for domain adaptation with the techniques presented in this survey. For instance, ordinary R-CNN could easily be converted to discrepancy based twin networks, since it is a standard CNN used for classification (used multiple times on one image). One could also force domain adaptation by adding a ReverseGrad header to the CNN (see section 4.4.1) or turn the CNN into an autoencoder as in section 4.6. In other words, all methods mentioned in this survey should hold. However, the R-CNN is not state-of-the-art, such that the more useful case, would be extending Fast R-CNN and Faster-RCNN. These are different, they rely on a feature map extractor, prepossessing the entire image (see section 2.3). They are CNNs chopped up into sub-networks, with internal algorithms dynamically deciding the flow of tensors. Domain adaptation would not be straightforward. However, some proposals will here be given.

For instance, ReverseGrad [36] relies on a feature map extractor. In DANN, the ReverseGrad layer back propagates into this extractor to maximize for domain confusion. Adding the ReverseGrad layer coupled with a domain discriminator as the third header to Fast/Faster-RCNN (next to the bounding box header and the object classifier header) should induce the same effect. However, since backpropagation does not flow into adjacent headers, only the feature map extractor will become domain confused, thus headers and RPN will not be affected by the domain confusion. However, this should not matter much if these overheads are shallow.

The approach proposed above was actually implemented using Faster-RCNN, by a team from the German university of ETH Zurich in Mars 2018 led by Chen [44], and it seemed to work. However, the RPN seemed to add some extra complexity since it relied on backpropagation from classification and bounding box headers to improve. Therefore, the team added a second ReverseGrad layer on top of these again to make sure the RPN could also receive updates (see figure 4.9 for architecture). A consistency regularization between the Domain Classifier and the stream following the second ReverseGrad would make sure the two streams wouldn't go in different gradient directions during training.

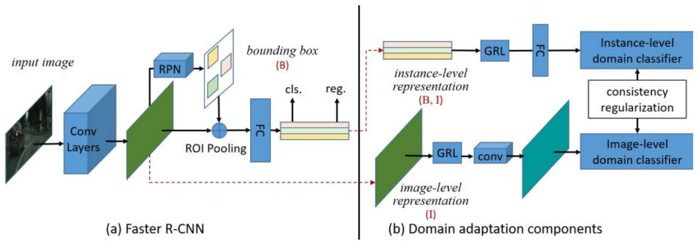

input image
Conv Layers
RPN
bounding box
(B)
cls.
reg.
ROI Pooling
FC
(a) Faster R-CNN
GRL
FC
Instance-level domain classifier
instance-level representation
(B, I)
consistency regularization
GRL
conv
Image-level domain classifier
image-level representation
(I)
(b) Domain adaptation components

Figure 4.9: A paper published during the work of this thesis show how adversarial and discrepancy methods can be combined to extend Faster-RCNN for DA. The extra components added to Faster-RCNN includes a ReverseGrad (called GRL in the drawing) and consistency regularization.

Fast/faster-RCNN and discrepancy methods

Extending Fast/Faster-RCNN for DA using discrepancy based methods is also something which should be considered. However, the outcome of such an architecture is difficult to predict due to Fast/Faster-RCNN's complicated pipeline. For starters, one would need to convert Fast/Faster-RCNN into a Siamese network with shared weights, or partially shared weights. One could follow the principle of DDC [38], using an adaptation layer and backpropagate its partial discrepancy measure, such as MMD or CORAL, into the rest of twin-network. However, this leads to a natural question to ask; where should one put the adaptation layer? Due to Fast/Faster-RCNNs multiple branching pipeline, no single layer comes to mind, thus using multiple adaptation layers seems like a natural choice. Here one could consider techniques such as BSW [26] and JAN [40], where one should calculate a discrepancy measure between multiple layers. Minimizing the discrepancy between multiple layers will make sure these layers maximize domain confusion and we would have a good approach for domain adaptation. However, the twin-architecture requires unison training of source examples and target examples. Generally speaking, for classification this would not be a problem, as we here expect objects in images to be centered. However, in object detection, objects are spread around in the image. Hence, taking the MMD or CORAL between two streams of features could represent a problem in how Fast/Faster-RCNN relies on local information, some areas would contain overlapping objects, but others not. In other words, if we were to measure the domain discrepancy on some layers, a large discrepancy could simply be attributed to some objects not lining up properly to match its counterpart stream. To get around this problem, one could follow the principles from Chen [44], where we divide the problem into two parts, *instance-level* and *image-level*. Image-level represents domain adaptation in the feature-extractor, where we only consider learning invariant features which are attributed to image style, illumination etc. Instance-level would be located in headers and RPN on Fast/Faster-RCNN and represents domain adaptation in features attributed to instances/objects we want to detect. Hence, we could say that shared weights (or partially shared weights) in the feature extractor would help domain adaptation only for image-level features, and domain invariant features on instance-level would only be done with shared-weights in headers. The architecture of the here proposed network, inspired by discrepancy methods and Chen[44], can be seen in figure 4.10 below.

In an unsupervised fashion, it is not certain that the target stream RPN (or Selective Search in Fast-RCNN), would only output true RoI positives. Here, clever methods need to be invented. It could be that true positive regions are more likely to have a high confidence, such that selecting only high confidence regions for discrepancy measures can get a good discrepancy measure. If so, it is reasonable to think that the target stream would need to be initialized from a pre-trained source stream in order to start handing good RoI to the instance-level discrepancy measure. To further combat single false positives, calculating instance-level discrepancies between source and target, can be averaged over batches.

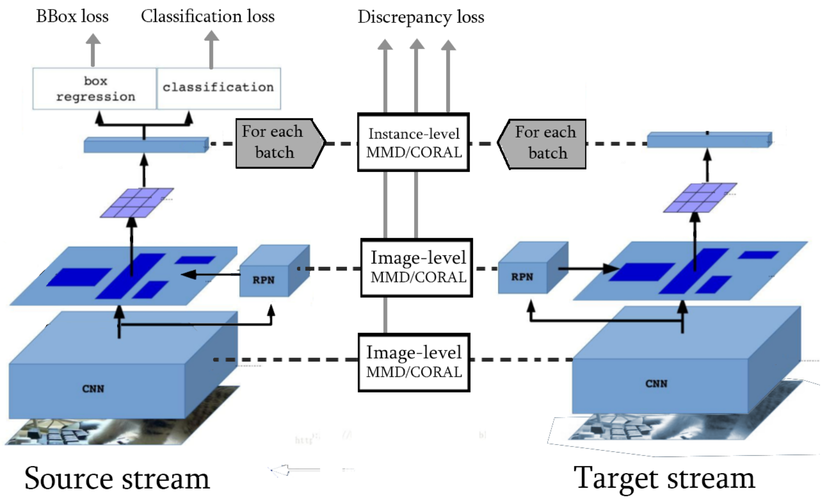


Figure 4.10: We propose a network inspired by discrepancy methods in DA, where the MMD or CORAL will be calculated between two streams (Siamese network) in Faster-RCNN. The discrepancy measure can be calculated between source-stream and target-stream in the feature extractor (image-level), however since this is an object detection problem; instance-level features (used for classification and bbox regressor) needs to be calculated between RoIs. Since we can't know that the target stream has true positives when predicting RoIs, we propose calculating instance-level discrepancies batch-wise to rely on an average discrepancy measure. Furthermore, this architecture can also be used with Mask-RCNN since it is built upon Faster-RCNN. Image source: Modified [12]

Object Detection beyond R-CNN methods

Looking into other object detection architectures, such as YOLO [23] and SSD [45], it is reasonable to think that domain adaptation principles discussed in section 4.7.1 is more flexible, and can be adapted better to fit these than they could for Fast/Faster-RCNN. The reasoning behind this postulate is that these methods rely on a single CNN, here used multiple times in one image. These architectures divide each image into a grid and runs the CNN on each grid-cell and outputs multiple detections for each grid, some with high confidence, some with low. Using the same reasoning as with standard R-CNN, looking at each grid-cell as a classification task (only here with multiple outputs and regression boxes) since these CNNs does not contain any headers nor dynamic parts, extending them to fit most discrepancy and adversarial DA techniques should be straightforward. However, SSD has an architecture where consecutive layers generate multiple predictions, making it a bit more complex. Such connections should not disrupt domain adaptation using discrepancy methods or adversarial, but a deeper study of behavior is needed to be certain.

4.7.4 Instance segmentation

In terms of speed and accuracy, current state-of-the-art for solving instance segmentation is Mask-RCNN. Since Mask-RCNN builds on Faster-RCNN with a third header for generating masks, domain adaptation should be seen as with Faster-RCNN in the section 4.7.3 and 4.7.3. With adversarial methods, the only exception is that the extra mask header is a large CNN within itself, that is; using adversarial methods as seen in DANN [36], adding a discriminator layer with a ReverseGrad layer as an extra header to the feature extractor, would help, but not for learning domain invariant features in the mask header, specifically. This can be solved by placing an extra discriminator in the mask header. If we were also to follow Chen's DA method [44] used in Faster-RCNN, this would add up to three ReverseGrad layers in the same architecture. Alternatively, using discrepancy methods as in our proposed architecture for Faster-RCNN, the extra header would contribute to instance-level discrepancy measure (see the Instance-level box on top in figure 4.10).

4.7.5 Fine-tuning, style-transfer and data augmentation

Even though extending CNN models to fit discrepancy, adversarial or autoencoding reconstruction methods requires some extra CNN components, the extra complexity added is only active during training. In other words, deploying such extended models on target data would be considered the same model as it was before we added the extensions. We simply chop off the extra components such as target stream network, discriminator etc. then run the old CNN on the target, but now with weights which is domain invariant. However, even though such extra components is not necessary influencing the internal mechanisms of our desired model, in practice; designing and implementing such components can take time, and requires a deep understanding of the code used. If one were to use open-source architectures implemented with libraries not familiar to the programmer, doing domain adaptation with these techniques may become a hassle. Here, fine-tuning and data augmentation techniques represents an easier approach, as they can do domain adaptation end-to-end without really altering the inner workings of a CNN. They are important because with them domain adaptation can be achieved by all end-to-end trainable CNNs. If an architecture becomes too complicated for advanced DA extensions, fine-tuning and data augmentation is the natural go-to. It could also be demonstrated by Csurka [34], at least with style-transfer techniques added, they can still compete against state-of-the-art DA techniques, even though they are considered classic. However, standard fine-tuning still bring along some problems. In semi-supervised and few-shot learning, having a small amount of data may not be sufficient to represent the real target domain, and overfitting may become apparent. Also, without these techniques to learn domain invariant features, there is also the fear that fine-tuning on only real data may change the weights substantially enough to overfit to the new domain, then losing some performance in the source. Moreover, in some tasks, it is difficult to acquire real training data, such as segmentation, where manually labeling images may take hours to complete [13]. Here, techniques for unsupervised domain adaptation would be preferable. However, if such extra labeled training data already exists, or if efforts put into implementing unsupervised domain adaptation exceeds the efforts of labeling such real training data, one should consider using these classical methods. For starters, the problem of overfitting can be reduced by data augmentation. Yet, the extent on how data augmentation can "squeeze" performance out from the few labeled target dataset target is not known.

4.8 Literature study conclusion

Most methods looked at for domain adaptation, have a similar principle. They consist of learning domain invariant features, and/or learn mappings from target images to features which is invariant to the domain. This is done by maximizing domain confusion either through adversarial training using a domain discriminator, by minimizing the discrepancy between domains by a direct measure of feature dissimilarity (such as MMD or CORAL), or by forcing classifiers to use the same features as in an autoencoder. Classical approaches can be done using Fine-Tuning and Off-the-shelf, where style-transfer represents a new state-of-the-art way to generate target-domain-similar training examples. There are other techniques not mentioned in this study, such as weighing source examples similar to target examples by discrepancy measure, but these are the most common ones.

Chapter 5

Experiment

5.1 Experiment background

5.1.1 Motivation

Domain adaptation has been extensively studied on the problem of classification making research on other computer vision problems still a partially uncharted field. Classification can also be solved using classical CNNs which often has no dynamic connection, and training is often done by normal backpropagation. As CNNs get more and more advanced, there is a larger need for more general domain adaptation techniques. Mask-RCNN represents state-of-the-art within the problem of instance segmentation. This architecture is rather complicated, but it is a good candidate of study, as it a prime example of how breakthrough is progressed in deep learning nowadays; it consists of many architectures stacked on top of each other. It could well be that future neural networks will keep moving in this direction, adding more and more networks to already state-of-the-art solutions, a bit like Lego. This makes Mask-RCNN an ideal architecture for research purposes, as we can experimentally learn more about DA on other problems than just classification, and at the same time look into a network which seems to represent the way research is progressing.

Moreover, the problem of segmentation is also a very visual one. That is; patterns predicted by the CNN and stamped onto pictures by a mask, give us a direct intuitive insight into how the CNN ”thinks”. With a strong visual experience, *seeing* whats possible with simulated data can help the industry of fish farming make better decisions on digitization and machine learning.

5.1.2 Choice of DA methods

For research purposes, it would be an ideal study to implement advanced techniques such as ReverseGrad, Siamese networks with discrepancy measures and/or autoencoders. That is; we could try to extend Mask-RCNN as proposed in section 4.7.4. However, this requires time and deep knowledge on implementation details of our code. As it was decided to go with an open-source version of Mask-RCNN [46] and due to time constraints, modifying its architecture was not prioritized. Moreover, since fine-tuning techniques is not well studied in computer graphics vs real images specifically - and due to fine-tuning is so versatile, they were selected as the DA methods for this experiment. Another technique which might be interesting to test is fine-tuning by gradually opening up layers from a freezing state. Let us say we only fine-tune top layers, based on the findings from Chu [29] (see section 4.3.2), this is recommended when target data is scarce, but when more data is available, fine-tuning the whole network can be preferable. However, something which has not been investigated, is what happens if we try to combine them? The idea here is to take the best from freezing, and no-freezing. After observing such techniques being tried by [46], we will try to gradually open up layers from a frozen state during fine-tuning. Here we will start by opening up the top layer and end by opening up all layers for fine-tuning. The desired experiment would be to do this incrementally opening up one layer at the time, but in practice, we will split the whole network into three parts, *overheads*, *the feature extractor from the 4th convolutional layer and up*, and *the three first convolutional layers*. This means we will incrementally remove each part from a frozen state and fine-tune them.

5.1.3 Some thoughts on Mask-RCNN

Due to Mask-RCNN being a complicated architecture, training is not what you consider straightforward. Even though it is end-to-end, there is a lot of dynamic parts and a lot of dependencies not found in standard fashion CNNs such as AlexNet and VGG-16. For one, the RPN can be seen as a separate ”helping function”, not actually contributing to backpropagation for the main task once a RoI has been predicted (see the chapter on basic theory, section 2.3.4 for architecture diagram). Training the mask header would backpropagate from the mask down to the backbone feature map, only entering the area on the feature map where the RoI has been extracted. It would then further backpropagate from this RoI-area, and affect only neurons in the feature extractor which feeds into this RoI. This means that many of the last weights in the last layer of the feature map extractor will not be affected by classification, mask and bounding box, unless they are used as a part of a RoI proposed from the RPN during forward pass. Yet, even though the RPN is somewhat a separate ”helping function”, more and more accurate RoIs will be predicted over time, giving better and better proposals to overheads. Furthermore, it can be said that the feature extractor has many jobs. It needs to do learn to produce features which help the RPN, mask-header, bounding box header and classification at the same time. With all this in mind, how this complicated training pipeline affects gradual fine-tuning in a DA scenario, is not well understood and is therefore considered an interesting object of study.

5.1.4 Method overview

An implementation of Mask-RCNN based on Tensorflow and Keras, with ResNet-101 as the backbone, was acquired from an open source repository [46]. It was partially improved to train faster and modified to receive a specific type of ground truth masks. Using Mask-RCNN a model was trained from scratch using synthetic images generated by computer graphics from a fish farming simulator called SimSalma [47]. The task was to *detect and* *predict silhouettes/masks for salmon in a fish farming environment.* After having a model trained as a source model, DA methods could be applied to it using photos from real-world fish farms as representatives from the target domain. Labeling such images was extensive work and due to time constraints, only 19 images of good quality was produced. These were split in half in order to further increase the training set and test/validation set. With such a small number of training examples, we are on the edge of what we can call semi-supervised. Based on whats considered normal in literature [5] [48] [49] we can define the experiment as a few-shot DA problem. Domain adaptation would be attempted using the following methods:

- Regular fine-tuning on all layers
- fine-tuning all layers with data augmentation
- fine-tuning all layers with data augmentation (only affine operations)
- Gradual fine-tuning layers
- Gradual fine-tuning layers with data augmentation (only affine operations)

Additionally, DA was attempted using a style-transfer technique and additionally combined with fine-tuning.

- Unsupervised style-transfer (fine-tuning only on style-transferred images)
- Unsupervised style-transfer combined with few-shot fine-tuning on real-world-mix
- Unsupervised style-transfer combined with few-shot fine-tuning on real-world-mix w/ affine augmentation

All fine-tuning and DA jobs were trained between 5 hours and 3 days, but the number of epochs used on each stage was determined by early stopping due to overfitting always becoming apparent after extensive time. The performance was measured using the *mean average precision* (mAP). This means average precision is based on the official PASCAL VOC [50] used for object detection. However, it was further modified for use in an instance segmentation setting. In PASCAL VOC, a true positive detection is considered true if an overlap has occurred where the overlap between ground truth and a detection is true if the intersection of union larger than 0.5. Here, the intersection was modified to be defined as the shared area of two polygon-masks measured pixel by pixel. Also, instead of 11-point interpolation over average precision as used in PASCAL VOC, 101-points was used as it gives a more detailed mAP. For a full mathematical definition of mAP, 11-point interpolation and the modifications, we refer to appendix 8.4.

5.2 Method

5.2.1 SimSalma

In cooperation with Ocean Farm 1, Kongsberg Digital's simulator SimSalma is a result from a comprehensive study of salmon behavior from scientific papers published by Sintef and NTNU [47]. It is a virtual salmon cage, containing salmon which is capable of responding to different variables such as cage, oxygen, food, light, temperature, water - current and quality, with up to one million fish swimming and reacting to its environment in real time. Salmon will migrate to its natural preferred position within the virtual cage, such that images taken from the simulator will have a realistic setup. The simulator can be used with most cages but is currently modeled to Ocean Farm 1, with cage wall, tubes, and platforms present in the virtual space. The computer graphics pipeline is running on Kongsberg's own 3D-engine called Cogs, which is based on how modern game engines work. Though the engine is powerful, due to SimSalma currently being in beta-stage of development, 3D-graphics has yet to be optimized for complete photorealism. However, this will probably only affect this experiment in terms of producing a larger covariate shift from synthetic to real. Below can we see a model of the entire cage used in fish simulation (figure 5.1).

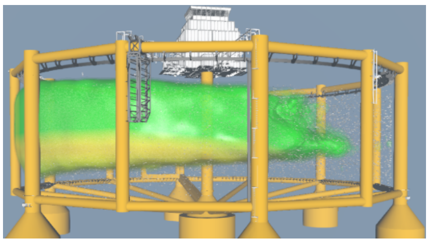


Figure 5.1: 3D model of Ocean Farm 1 farming cage used in SimSalma simulator. The green color is the density of fish steams affected by the simulation of the natural habitat.

5.2.2 Extracting data

SimSalma came with an API making the program accessible for real-time modification of parameters and variables by Python. A Python script was implemented - able to generate an infinite amount of pictures within the virtual water cage. This was achieved by programmable interacting with a virtual Cogs camera. When taking a new picture, the camera was translated to random positions within the virtual cage with random orientations. Furthermore, a script which could count fish from ground-truth images, made sure, that images not containing fish would be deleted. Also, due to a lack of GPU memory, Mask-RCNN was not able to train on more than 800 fish in one image - images beyond this threshold was removed. To achieve a greater variation in the simulated dataset, each image was generated with a different sea environment where the color of the sea could vary, as well as transparency and light intensity. This was done to simulate the variation of water quality in the real world cage of Ocean Farm 1, considering reports of algae could make the water green, less transparent and/or cloudy weather could make the water darker, thus the visible distance is reduced. Three base watercolors were chosen; green, blue and grey. After three days of running the script, 20,856 images were generated, as well as corresponding ground truth segmentations. Yet, the dataset was not ready for training, because additional post-processing was needed for making data fit our model. This involved reducing the size of images to 704x704 pixels and converting ground truth segmentations to a format Tensorflow could understand. The ground truths came out-of-box as a png-images, where each object had a silhouette filled with a unique color, including virtual tubes, cage-walls, sea bottom, and equipment. A script was implemented which removed these excess objects, as well as removing salmon ground truths too small (less than 6x6 pixels), and salmon too far away from the camera (not being visible in the corresponding render image). The resulting dataset contained approximately 15,000 images of salmon with corresponding ground truth masks (see images in figure 5.2 below to see a typical set of ground truths masks).

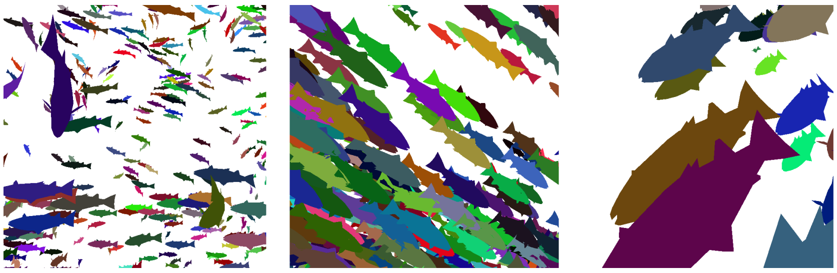


Figure 5.2: Post-processed ground truth masks generated from SimSalma

5.2.3 Dataset bias

The SimSalma dataset of 15,000 images is substantial in magnitude for deep learning applications, and probably won't allow for overfitting, that is; due to the size of the training set, the model is not likely to start memorizing the training dataset. However, as Rajpura[20] has shown by experimentation, synthetic dataset bias can actually appear if training on too much synthetic data. Here, we shall point out some observations about the differences between the real and synthetic that can impact transfer learning / domain adaptation due to dataset bias:

- *Computer graphic bias*
  Is computer graphics too perfect? Real images can often come with a shaky camera (which blurs the image), static noise from dark scenes and various lens flare effects produced in the optics of a real camera. There is also the fear that computer graphics can paint a picture of reality which is too general, especially real-time 3D-engines which use a simple computational pipeline for fast realism. Such assumptions could be; that light reflects only in a few ways of textures, that anti-aliasing is similar on all objects, that textures are often repeated to reduce memory, and that shadows are often simple. The fear is that machine learning approaches, in particular, deep learning, can feed on these truths, learn patterns, and produce features only applicable to computer graphics.

- *Model bias*
  The simulated dataset only contains one type of salmon - different sizes, yet with the same textures and same general shapes. Also, even using a variety of parameters such as different sea colors and light intensities, the simulation was not done with different amounts of sea turbulence, sky backgrounds or angles of incoming light from the sun.

- *Compression bias*
  Images and video come in different compression techniques for minimal storing and fast uploading/downloading. Different compression techniques often create different artifacts. In particular, video from the target domain is stored in mp4 with H.264 compression, which contains less artifact on moving objects than for instance the background. This is due to how the compression algorithm works. This could be identified as a potential problem during domain adaptation due to the simulated training data not containing such artifacts.

To counter all these potential dataset biases the images were heavily augmented with various filters and effects. More details on this will be given in the next section.

5.2.4 Data augmentation

70% of the data was augmented with 2 to 4 filters and effects for each image, producing a strong regularization on the CNN which reduces dataset bias, easing DA. The set of filters and effects were regularly switching between *contrast and gamma* to simulate overexposure, *blur* to simulate camera shaking, *emboss* to simulate different shadow angles, *darkening* to simulate night and deep waters conditions, *add to hue and saturation* for a deeper range of colors, but also others effects such as *sharpening* and *inverting of R, G or B channels*.

Since all pictures/video from the real world has a small amount of grain to it, *noise* was added to 50% of the data. Lastly, as a general deep learning technique, random black spots, white spots and noisy squares was added to force the network to make more neurons / cooperating across layers. The magnitude of each filter and effect was picked out randomly from a set of ranges which was continuous. Augmentation was done using the well-known library called Imgaug [51]. A random subset from the SimSalma dataset with a random subset of data augmentation can be seen in figure 5.3.

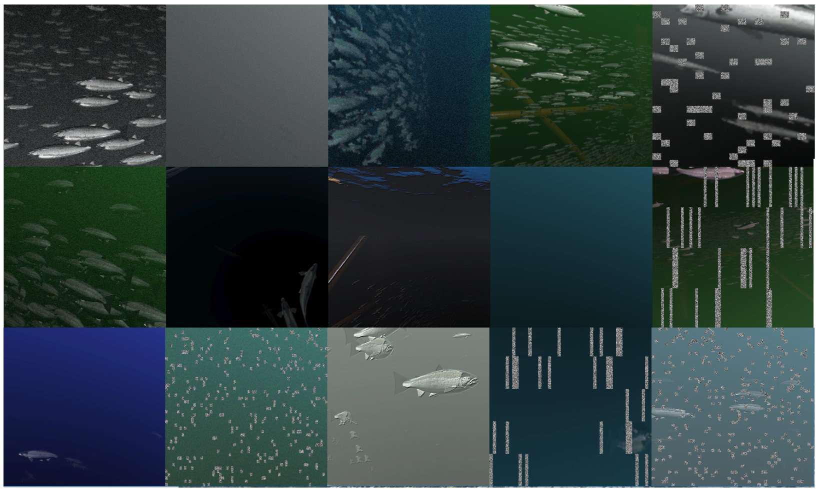


Figure 5.3: Data augmentation performed on SimSalma / how data would look like before it was fed into Mask-RCNN.

5.2.5 Training

The simulated data was split into a training set, a validation set, and a test set, with the ratio of 14800:100:100. The reason for a small training and validation set was due to time constraints on testing. However, since each image contained 239 fish on average, and mAP would be calculated instance wise over the whole dataset, statistical significance was still kept. Training was done over a period of 40 days on Geforce GTX 1080, with an epoch of 76 and batch size of 1. After this point, the network would have seen 1,140,000 images, where statistically 98.5% images would be more or less unique due to data augmentation. For optimization, similar to the paper on Mask RCNN, stochastic gradient descent was used with a learning rate of 0.001 and momentum of 0.9. Furthermore, a weight decay regularization term set to 0.0001 was used. Due to Mask-RCNNs complicated pipeline, during training five loss functions needed to be monitored:

- *RPN bbox loss.* The Region Proposal Networks bounding box refinement loss.
- *RPN classification loss*. The Region Proposal Network classification of ”object” or not ”object”.
- *Mask header loss*. The loss describing the mask header ability to create a good mask from region proposals.
- *Classification header loss*. The loss describing the classification of ”salmon” or ”background”.
- *Bbox header loss*. The loss of bounding box refinement of edges around fish.

While the total mean loss of all validation tests descended rapidly and converged after a few days (see figure 5.4 for mean training graphs), the classification header loss actually increased the first few days. However, after some days, it started to decrease slowly, linearly over a month. It's hard to tell exactly why this was the case, but it could be explained by the fact that Mask-RCNN divided tasks into parts, here some parts of the network could lag behind. In the beginning, training on random RoIs would make it easy to classify RoI extracted from empty space as ”background”. However, as ResNet-101 starts to feed in RoIs with fish, it can't keep up, and we see an increased loss. However, a steady improvement appeared as the classifier started to recover. Furthermore, a visual inspection was made on images, and with good results, the obscurity was not of concern. The resulting model trained completely on simulated data will be referenced as *Synthetic-model*, but also as the source model.

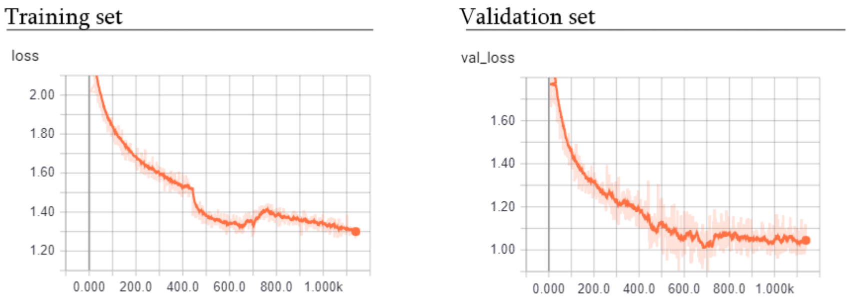


Figure 5.4: Training of Synthetic-model over time. The small dip seen in the middle was a change in training strategy (the amounts of fish used in each image), but it was later changed back due to not showing any effect of faster training.

5.2.6 Making real-world data

The production of segmentations / masked images for deep learning is one of the most laborious tasks within computer vision [13]. As a testimony to this fact, the production of a real-world test set held an average working time of one hour per image. In total, 80 HD images were masked, where 12 images came from Ocean Farm 1, and 68 frames from an eight-minute-long video recording in one of Lerøy Midts land-based cages. Five of the 68 frames from Lerøy Midt, was taken out randomly at different time codes, while 63 was not random, but rather a sequential two-second none-interrupting video. The labeling process was carried out in two separate manners. The first; painting silhouettes around fish and filling them with unique colors in Photoshop, a process which was done on all still frames. However, ground truth masks for the video was created using After Effects. After masks were created in After Effects, images were saved as still frames. The whole process of labeling 80 images took about two full working weeks to complete. From this point forward, a combination of Ocean Farm 1 and Lerøy Midt images with corresponding ground truths will be referred to as the *real-world dataset* (see the top of figure 5.5).

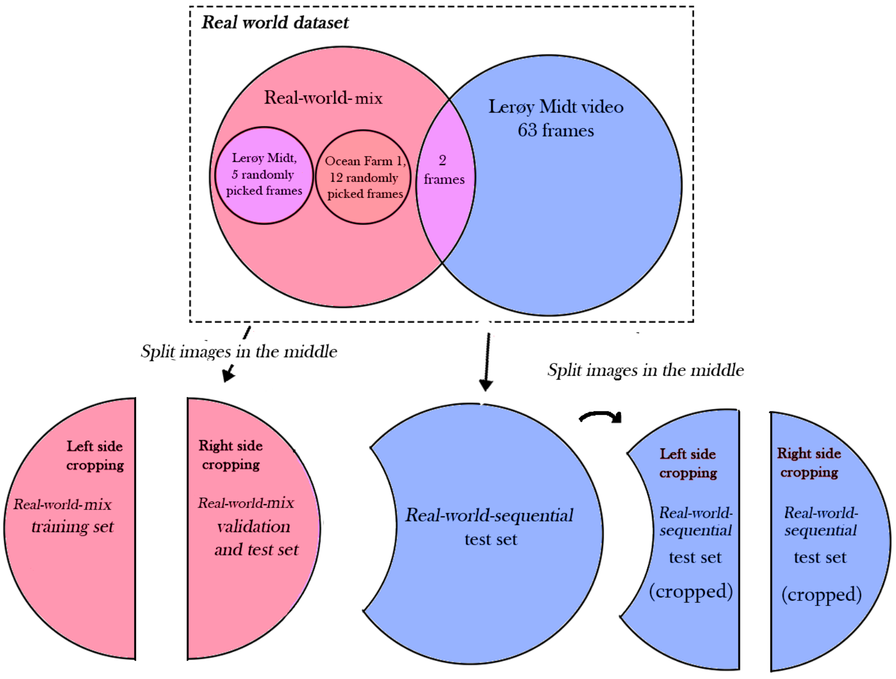
5.2.7 Details about real-world dataset

Due to having a small amount of real-world data, some creativity was needed in order to bring the most of the data. This will here be elaborated.

For training (fine-tuning), it was important not to mix Ocean Farm 1 video-frames and Lerøy Midt images too much. This is justified by the strong correlation between frames in the two-second video, which will probably make any model overfit to the set of salmon poses represented in it. Therefore, a choice was made to have a training set containing only two images from the sequential footage; one frame taken from the end, and one from the beginning, thinking that these are the most different from the rest of the dataset (see the purple field representing union in figure 5.5). Extracting the first and last image would give us training examples from the sequential footage, yet being somewhat distinguished from the rest 61 test images. As an alternative, one could leave out training examples from the sequential video, but then having the model not trained on a single example could make a strong dataset bias towards the Ocean Farm 1 environment. At this point, the training dataset would contain 12 Ocean Farm 1 images, 5 Lerøy Midt images and 2 Lerøy Midt video images (see red circle in figure 5.5). We called this dataset *Real-world-mix.*

Real-world-mix, containing only 19 images, was too small to split into a validation/test set and a training set. To double the real-world-mix dataset, the fact that these images were HD was exploited. The implementation of Mask-RCNN used in this thesis was set to an input of 704x704 pixels, forcing each image to either be downscaled or be cropped. Instead of downscaling HD images each time we feed them into Mask-RCNN, it was chosen to crop each image in two - creating twice the amount of data. The left-side was used for training and the right side for testing (see ”split images in middle” on the red circle in figure5.5.

These splits are justified by the following: As each fish on either side of the split is unique, training and testing/validation data would be considered different. However, there is a strong correlation between the two cropped images due to similar light conditions, sea color, and transparency. There is the fear that a strong covariance between the surroundings could make test results on the validation set blind to detecting overfitting to such environment-features. That is; there is a strong dataset bias towards sea color, transparency, fish size, and fish orientation, and since both sides (test set validation set) is similar, it’s difficult to detect. If such dataset bias will occur, this will be countered by a control model *pure-target-model*, more on this later. Moreover, since such features are very high-level abstractions of features, we can still hold to the idea that the splitting only produces a minimal bias between the training and test/validation sets.



Real world dataset
Real-world- mix
Lerøy Midt video 63 frames
Lerøy Midt, 5 randomly picked frames
Ocean Farm 1, 12 randomly picked frames
2 frames
Split images in the middle
Split images in the middle
Left side cropping
Real-world-mix training set
Right side cropping
Real-world-mix validation and test set
Real-world-sequential test set
Left side cropping
Real-world-sequential test set (cropped)
Right side cropping
Real-world-sequential test set (cropped)

Figure 5.5: How real-world datasets are catalogued and categorized. 12 Ocean Farm 1 images, 5 Lerøy Midt images and 2 video frames are put into a mix called *real-world-mix* (red). This is further divided into a training set and a test/validation set by splitting images in the middle. On the other hand, two frames are taken out from Lerøy Midts video (blue), and the rest of the 61 frames become *real-world-sequential*. This is further split in the middle, doubling the dataset - and become *real-world-sequential (cropped)*.

5.2.8 Testing on real-world

Performance testing of DA methods was always done in three parts:

Real-world-mix test set) Random photographs from Ocean Farm 1, but containing some few frames from Lerøy Midt. (Testing will be done on the right side sub-images of Real-world-mix). In total 17 images. All images from the Real-world-mix test set can be seen in Appendix 8.6 (Note, the experiment has been conducted with mask detections applied to all images from the leading DA method.)

Real-world-sequential test set) A dataset where each image is a frame in a two-second long sequential video taken from Lerøy Midt. Every frame is 1920x1080. In total 61 images.

Real-world-sequential (cropped) test set The real-world-sequential frames from the test set above was split in the middle to double the amount frames. Here, every cropped frame would become 704x704. In total 61*2 images (see ”split in the middle” on the blue circle in figure 5.5).

(Details on the origin of these datasets in (5.2.6), and the justification of choices can bee read about in (5.2.7))

Be in mind that due to real-world-sequential (both cropped and uncropped) containing images from just a two-second video, there would be a sampling bias and a strong dataset bias as the result. Yet it provides information on how sensitive the model is to small differences in salmon poses, occluded parts of salmon and overfitting.

Due to lack of data, in the case of the real-world-mix dataset, test data and validation data will be the same sample. This will impact the final test results when using the technique of early stopping. Here, there is no guarantee that the selected epoch is not suffering from *data dredging / data fishing*. This is where someone continuously tests a large amount of hypothesis, only to find one that correlates good to a prediction, but due to it being cherry picked. During early stopping, one often select the model corresponding to a bottom critical point in the error/epoch-plot. However, when selecting a bottom point, it is not certain that the bottom point you selected represents the true performance of your model, or if it is better than a test set, due to data dredging. To combat this effect, testing on the real-world-mix validation set will also be done to adjacent epochs, taking the mean of three epochs, smoothing the result.

5.2.9 Making domain testing similar

For testing on the synthetic domain, 100 SimSalma images were used. It became clear during the work of this experiment, that the 17 images used for testing real world (real-world-mix) were weighted differently than the SimSalma testing set. Each image contained on average 75 fish with a standard deviation of 80, while SimSalma had 239 fish on average with a standard deviation of 410. Another observation was that Mask/Faster-RCNN is not designed with the purpose of detecting hundreds of objects at the time, even though it can train on 800. Supporting this claim is the fact that most instances detected in synthetic images containing a small amount of fish, but in pictures containing hundreds of fish, many fish was left out. The SimSalma test set was therefore weighted with examples such that standard deviation and mean was equal to the real world test set. This will help us compare progress on the target domain using DA methods.

5.2.10 Control models

Synthetic-model works as control model/baseline for verifying that DA actually takes place, therefore synthetic-model was tested on the real world datasets as well as DA models. Also, synthetic-model was also tested on the 100 simulated images to give an upper line for what we should expect from a perfect DA model. Furthermore, another control model was created. In a few-shot scenario, training a model from scratch on 19 images is difficult. However, such a model was attempted using affine data augmentation such as rotate, shear, translate and scaling to extend the dataset. It was successful. This model will be referred to as *pure-target-model* and will be a baseline for how we should rate DA models.

5.2.11 Fine-tuning with/without augmentation

Fine-tuning was attempted using synthetic-model as initialization, and real-world-mix training set on all layers. Moreover, in addition to regular fine-tuning, fine-tuning using data augmentation was attempted. It was done twice using two different approaches. Here, the motivation is to see how much information can be "squeezed" out of 19 images with data augmentation in order to improve the deployment of synthetic-model in the wild. Data augmentation is considered standard practice in deep learning, but using classical data augmentation methods purely for few-shot DA scenarios is not well studied in the literature. Details for the two approaches will here be given:

- *Heavy augmenter)* The first data augmenter is similar to the one used by training synthetic-model, as it contains a lot of colors, contrast, blur, sharpen operations, but also affine operations such as rotate, shear, translate and scaling was added as well. During fine-tuning, 80% of all images were augmented.
- *Affine augmenter* The second augmenter only contained affine operations. The suspicion behind this choice was that too much augmentation might act as a too strong regularizer, making the model underfit to unseen target test data. Affine operations only transform the fish shape and position but do not affect the environment.

5.2.12 Gradual fine-tuning

Gradual fine-tuning was done in three stages.

1. First, the feature extractor was frozen, while all layers following; such as classification, bounding box, mask header and RPN, were opened for fine-tuning.
2. At the second stage, all layers following the fourth layer in ResNet-101 was opened to fine-tuning.
3. At the third stage, all layers were opened to fine-tuning, this time using $1/10$ of the original learning rate.

The number of epochs used on each stage was determined by early stopping due to overfitting always becoming apparent after extensive time. For training, the real-world-mix training set was used, but the gradual fine-tuning experiment was further attempted in two versions. In the first version, all 19 cropped images were used. In the second version, data was heavily augmented using affine operations as the one described in 5.2.11. This was to generate complementary models for comparison between non-gradual fine-tuning models.

5.2.13 Style-transfer

The last experiment attempted in this master thesis, was a style-transfer DA technique using a style-transfer GAN from an open-source repository [52]. This style-transfer network was designed for transferring a style from one image to another, without extensive extra training to learn features on the input style-image. This meant that one single image containing ground truth and salmon instances from the SimSalma training dataset could be synthesized with a style from an image from the target domain in just 84 seconds (on a Geforce 1080). However, the style-transfer GAN was primarily designed and trained for transferring art styles, such as used in paintings. This might have an impact on the performance. Furthermore, 500 simulated images were randomly picked from the SimSalma dataset of 14800 training examples (see images below 5.6), and style-transferred to look ”real” by a randomly coupling it with a style taken from a random frame in Ocean Farm 1 or Lerøy Midts videoes. This new dataset of 500 images would be used as a training set for further DA methods. The first style-transfer DA method involved initializing a model using weights from synthetic-model, then fine-tuning on these 500 images. Since labeled target examples were not involved in this process (except unlabeled used for generating styles), this fine-tuning would be considered unsupervised DA. We refer to the model produced by this DA method as *unsupervised-style-transfer-model*. The second style-transfer DA method attempted, involved initializing weights with unsupervised-style-transfer-model and further fine-tune on real-world-mix training dataset (few-shot DA). We will refer to this model as *few-shot-style-transfer-model*. The last method was similar to the few-shot-style-transfer-model, but the real-world-mix training set would in this occasion be augmented using affine augmenter (see details on affine augmenter in section 5.2.11). This model will be referred to as *affine-few-shot-style-transfer-model*.

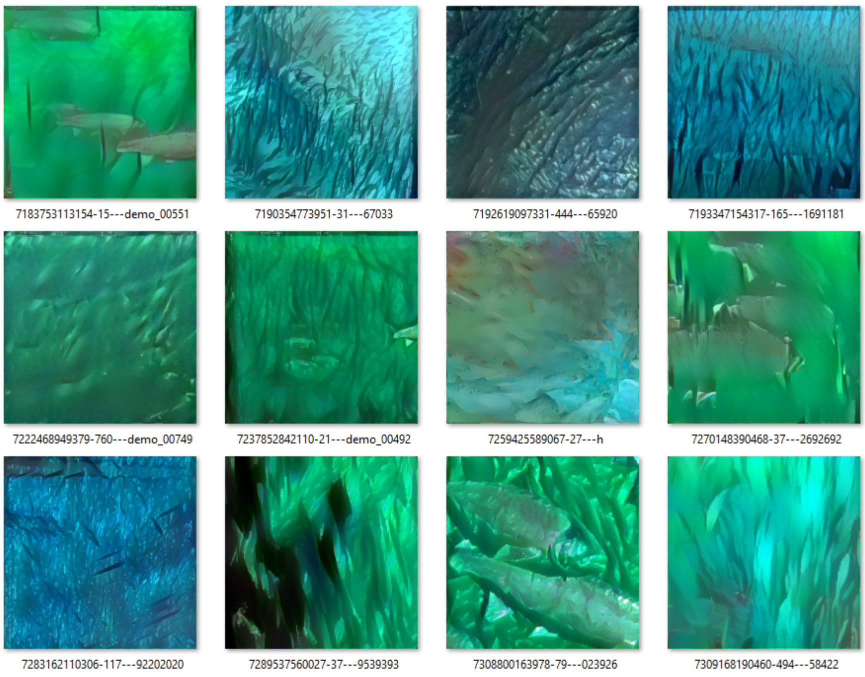


Figure 5.6: Hybrid images synthesized by a style-transfer GAN with ground truths and fish positions copied from the source domain, and styles from the real world. As seen in pictures, confusing patterns is apparent and might conflict with DA.

Chapter 6

Results

In this chapter, results will be presented and validated. At the same time, there will also be some discussion on observations made, during fine-tuning and setup of the experiment. All performance results can be seen in the table 6.1 below and style-transfer results at the end of this chapter (see table 6.3).

	Performance measured in mAP(%) (mean Average Precision)			
	Synthetic test set (source domain)	Real-world test set (target domain)		
	SimSalma (100 images)	**Real-world-mix test set**	**Real-world-sequential (cropped)**	**Real-world-sequential (uncropped)**
Control models (no domain adaptation)				
Synthetic-model	55.75	9.37	10.38	10.19
Pure-target-model	0.02	24.97	34.06	2.36
DA technique				
Regular Fine-tuning (using 19 real images)	**56.83**	27.48	31.36	16.35
Fine-tuning w/ data augmentation	44.34	14.12	13.00	9.32
Fine-tuning w/ affine data aug.	24.57	**41.85**	**48.84**	**21.37**
Gradually fine-tuning (using 19 real images)	55.25	28.51	33.86	16.58
Gradually fine-tuning w/ affine aug.	36.91	36.45	42.71	14.40

Table 6.1: Results on all few-shot DA techniques using classical fine-tuning. Synthetic-model is only trained on 15,000 synthetic images. Pure-target-model is only trained on target domain (using affine data augmentation). All DA techniques are pre-trained on synthetic images, or in other words; weight-initialized with Synthetic-model, then fine-tuned on real images. Numbers in **bold** represent leading scores on datasets, relative to other DA techniques (across rows). The best performance achievable by DA on real data was ”Fine-tuning w/ affine data aug.”, with 41.85 mAP(%) (using Real-world-mix as reference). See section 5.2.8 for description on datasets *Real-world-mix*, *Real-world-sequential (cropped)* and *Real-world-sequential (uncropped)*. Furthermore, the first DA technique *Regular fine-tuning* achieves 27.48 mAP(%) in the real world domain (using Real-world-mix as reference) which is quite a lot considering fine-tuning was only done with 19 real-world images. At the same time, it seems that Regular Fine-tuning (using 19 real images) actually increases Source Domain performance.
Results also seem to suggest that data augmentation is good for the real-world test set but reduce the source domain performance. Moreover, gradually fine-tuning gives mixed results.

6.1 Synthetic-model performance

The performance of the source model trained completely in the synthetic domain scored 55.75 mAP(%) when tested on the synthetic (see *Synthetic-model* in table 6.1). This was good, but excessive data augmentation was necessary on the synthetic dataset to reduce dataset bias and allow for a more smooth transition to the target domain. A small extra performance could probably be gained in the synthetic domain, using a final fine-tuning on the synthetic dataset using no data augmentation, but at the expense of DA as it would create more dataset bias. Moreover, it is believed that additional performance could be gained by creating a better clipping algorithm. Pictures of Synthetic-model performance in the synthetic domain can be seen in figure 6.1.

As expected, due to a covariate shift, the synthetic-model have trouble performing on the real-world dataset. Performance drops by a staggering 83% percent compared to how synthetic-model does in its source domain (comparing *Synthetic-model* on *Real-world-mix* to SimSalma in table 6.1). It would appear that dataset bias plays a big role, even though 3D fish models are similar to real fish.

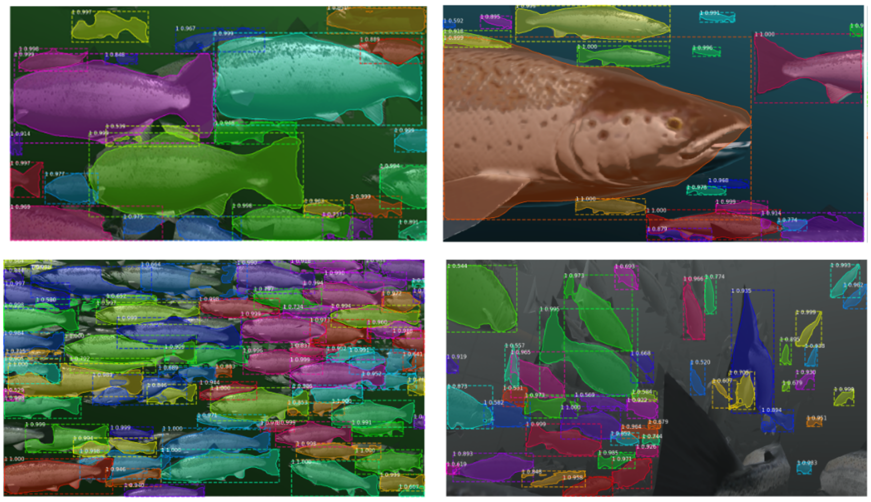


Figure 6.1: Pre-trained model (Synthetic-model) performance on its synthetic home-domain. Model perform 55.75 mAP (%) on this domain (ref table 6.1). Most false negatives are located in crowded areas of fish where the salmon is not seen in profile. More images of synthetic-model on synthetic images can be seen in Appendix 8.5

6.2 Pure-target model

Training a model from scratch on just 19 images using affine data augmentation produce some surprising results, it does 24.97 mAP(%) on real-world-mix and 34.06 mAP(%) on real-world-sequential (cropped) (see *Pure-target model* in table 6.1). However, the absent performance on real-world-sequential (none-cropped), and the synthetic dataset shows a very strong dataset bias, such that, this control model has probably only learned features associated with a few camera angles and fish sizes. However, as the model will only be used for control, it works well as a basis to interpret results on real-world-sequential (cropped) and real-world-mix.

6.3 DA techniques - results

6.3.1 Regular Fine-Tuning

Though 19 images represent a very small dataset, fine-tuning the whole network with these images increased the performance considerably (see *Regular Fine-tuning (using 19 real images)* in table 6.1). Going from the source model to target fine-tuned increased the performance almost by a three-fold (from 9.37 mAP(%) to 27.48 mAP(%) on real-world-mix). This is consistent with the results discovered in the project thesis [22], where only 10 images increased the synthetic-model performance *in the wild* by 73%. Seen in relation to how the synthetic-model does on its own test-set 55.76 mAP(%), a performance of 27.48 mAP(%) in the real means that the fine-tuned model has not fully reached its potential, though it is not bad. At the same time, we can observe that there is no large gap in performance between the source model and the fine-tuned model tested on the synthetic test set, it has actually been improved, suggesting the model has learned domain invariant features to reduce the covariate shift. The small improvement of 1 mAP(%) might be attributed to the pre-training (Synthetic-model) being heavily regularized with data augmentation, reducing underfitting by introducing sharp, realistic-looking, images. Furthermore, there is also a good domain adaptation on the real-world-sequential dataset (cropped), but the extent at which this is due to overfitting to unique picture features is not clear since testing on real-world-sequential uncropped version only shows a minor boost (from 10.19 mAP(%) to 16.35 mAP(%)). However, all models in this experiment, tested on the uncropped real-world-sequential does poorly in general. Since all instances in real-world-sequential have a strong covariant relationship, the poor performance on the uncropped version could also be attributed to sampling bias and the fact that smaller fish in these examples haven't been seen by our model. Moreover, compared to pure-target-model, the fine-tuned model shows a slight performance gain on real-world-mix but does worse on real-world-sequential. Since real-world-mix is more reliable and show some gain on real-world-mix, this substantiates that synthetic data combined with real is better than on only real data, at least in a few-shot scenario. During fine-tuning, overfitting started after seeing the dataset 23 times. Performance of regular fine-tuning can be viewed in images below in figure 6.2

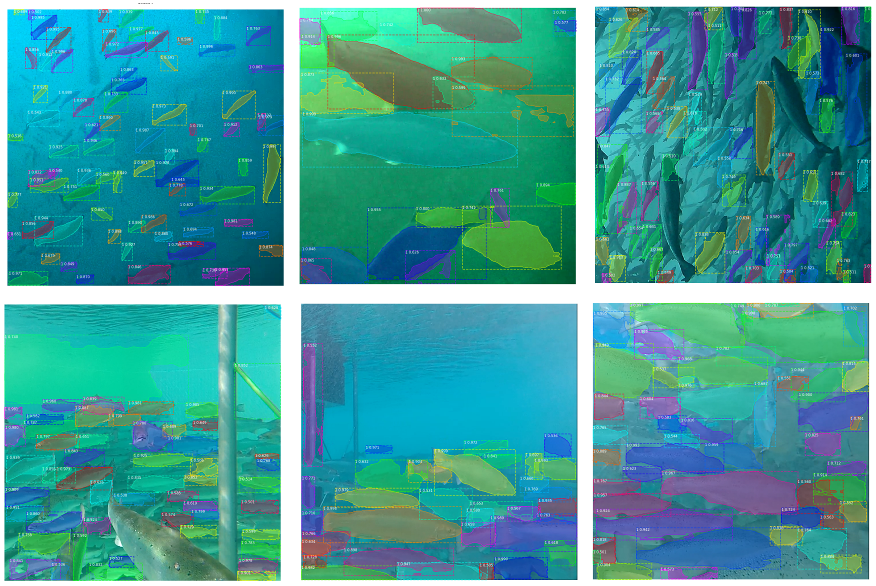


Figure 6.2: Regular fine-tuning performance on photos from real-world dataset (colors is model mask detections). On top we can see pictures taken from Ocean Farm 1, while at the bottom we can see pictures from Lerøy Midt. Model performs good with 27.48 mAP (%) (ref table 6.1), but have a hard time detecting fish in clusters.

6.3.2 Fine-tuning w/ data augmentation

In the case of fine-tuning all layers with data augmentation (see *Fine-tuning w/ data augmentation* in table 6.1), a large gap in performance between heavy augmenter and affine augmenter became apparent. fine-tuning with color, contrast, blur/sharpen, and affine operations resulted in an underfitting, doing even worse than pure-target-model on real-world-mix and real-world-sequential (cropped), and worse than source model / synthetic-model on real-world-sequential and synthetic data. This is likely due to heavy augmentation producing training data so dissimilar to the real target domain, that it over-regularize and underfitting occurs.

6.3.3 Fine-tuning w/ affine data aug

Using too much augmentation was probably not the case for the affine augmenter (see *Fine-tuning w/ affine data aug* in table 6.1), as it outperformed all other methods in this experiment. With 41.8 mAP(%) in the target real domain, the model even outperforms regular fine-tuning. It is reasonable to believe that a lot of this performance must have come from pre-training on source domain, as pure-target-model which has been purely trained with the same images, and the same type of data augmentation, can't compete in performance.

However, we can see a large set-back when testing on the synthetic domain which indicates that the model has lost some source features and started to gain a new bias towards the target domain. This is to be expected when training for a long time on a new dataset. Fine-tuning was here done for 600 epochs before overfitting started to become apparent. Compared to 23 epoch which was the case for fine-tuning without data augmentation, we can safely say that data augmentation catalyzed extraction of more information from the small dataset of 19 images.

Fine-tuning w/ affine data aug performance on the Real-world-mix test set is visualized, and can be seen in Appendix 8.6.

6.3.4 Gradual fine-tuning

Gradually fine-tuning (using 19 images)

Gradually fine-tuning on 19 images (without data augmentation) showed a slight improvement over regular fine-tuning (see *Gradually fine-tuning (using 19 real images)* in table 6.1). However, the improvement is minuscule and hence it is difficult to know if it is purely caused by the gradual training scheme. Looking at the graph in figure 6.3, most of the performance gain is located on training overheads, which is consistent with Yosinski findings that early layers are general (section 4.3.1 in literature study). During the process of fine-tuning, there was also the appearance of overfitting at early stages. Overfitting became apparent already at the 8. epoch when training headers, it became apparent at the 1. epoch after opening up layers following layer number 4 and appeared after the 10. epoch when opening all layers. However, all overfitting was expected due to the low number of training examples, but interestingly it seems to start earlier when doing gradual fine-tuning. Furthermore, as with regular fine-tuning, testing the gradual fine-tuned model on synthetic data showed no large decrease (showing 55.25 mAP(%)), suggesting domain invariant features is learned as with regular fine-tuning.

Gradually fine-tuning w/ affine augmentation

As for *gradual fine-tuning* using affine augmenter, the relative performance in table 6.1 tell a different story (see *Gradually fine-tuning w/ affine aug*). Though the model does better than *regular fine-tuning*, this can simply be attributed to the fact that we use affine augmentation which has been shown to be quite effective for this particular DA scenario. If we compare the method with its none-gradual counterpart, *regular fine-tuning using affine augmentation*, we see that the current DA model perform 13-30 mAP(%) worse across the real-world datasets. This is somewhat mysterious and will be discussed in the next chapter. We can further note that *Gradually fine-tuning w/ affine aug.* maintain performance on synthetic data better than its non-gradual counterpart regular *fine-tuning using w/ affine aug.*.

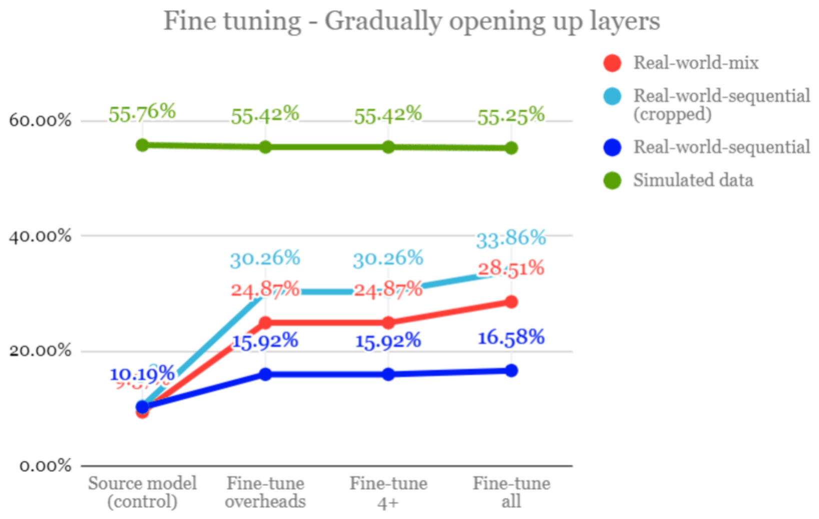


Figure 6.3: Gradual fine-tuning over time, opening up new sections from freeze state to fine-tuning at each unit on the x-axis.

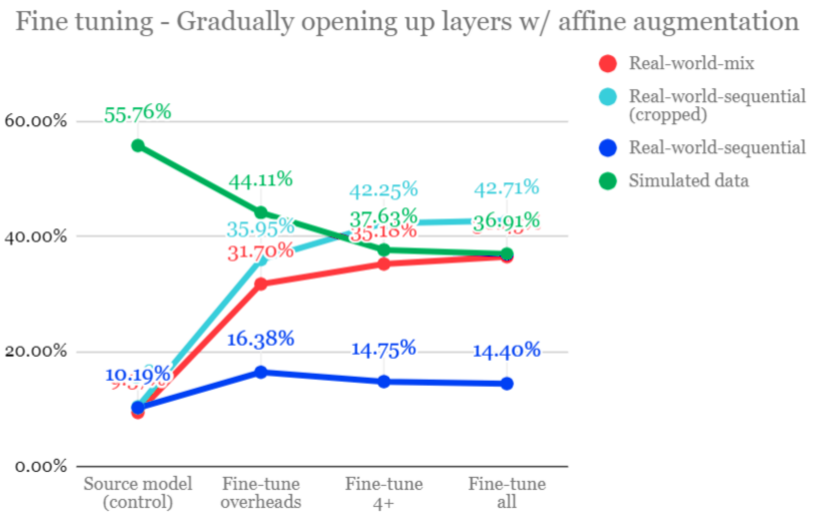


Figure 6.4: Gradual fine-tuning over time using affine augmentation. Opening up new sections from freeze state to fine-tuning at each unit on the x-axis.

6.3.5 Style-transfer

The performance of our attempted style-transfer DA was not good. Comparing Unsupervised-Style-Transfer-Model (see table below 6.2) to Synthetic-model (line one in table 6.1), we see a large setback on the development of the model after fine-tuning on the style-transfer dataset. During fine-tuning, performance on real-world-mix was constantly monitored, and there seemed to be no convergence, only deviation from the target domain. At one point there was a slight dip in the loss. Fine-tuning was stopped at this point and weights used as Unsupervised-Style-Transfer-Model, which naturally couldn't perform well. However, the experiment would continue, perhaps Unsupervised-Style-Transfer-Model would become better if we fine-tuned it using 19 images from real-world-mix, that is testing Few-shot-style-transfer-model. However, during fine-tuning, overfitting became apparent from the first epoch, suggesting that Unsupervised-Style-Transfer-Model had gotten lost in weight-space, and a "small push" in the correct direction couldn't fix it. Finally, using affine augmentation could perhaps give it a larger push without overfitting at such an early stage. However, though Affine-few-shot-style-transfer-model did quite good compared to methods previously attempted in this thesis, comparing it to *Fine-tuning w/ affine aug* seen in table 6.1, we see that no extra boost was added from the style-transfer dataset.

The low performance on our attempted style-transfer methods is not surprising. We can identify the following problems which might have contributed to a setback:

- *Artistic style*) The GAN used to produce images was used for art, such that training examples didn't look completely realistic, but had a hint of "paint-style" to it.
- *GAN not tuned for object detection*) The architecture of the GAN was probably not

optimized to identify objects in the image and assigning a unique style to unique objects. This became apparent by visually inspecting the style-transfer dataset, that many of the salmon in the realistic style-reference image was interpreted as ”brush strokes”. The result was ”salmon inside ground truth synthetic salmons”, and salmons decorated in the background which had no ground truth referenced to them. It should be clear that this would massively confuse a CNN, as such ”false detections” would push gradients towards interpreting good salmon features really as background.

These findings suggest that style-transfer techniques are best for classification tasks, as no object detection is needed and ”brush strokes” have a harder time fooling the classifier to make false positives.

	Performance measured in mAP(%) (mean Average Precision)			
	Synthetic test set (source domain)	Real-world test set (target domain)		
	SimSalma (100 images)	**Real-world-mix test set**	**Real-world-sequential (cropped)**	**Real-world-sequential (uncropped)**
Style-Transfer DA technique				
Unsupervised-style-transfer-model	4.07	5.59	2.81	39.76
Few-shot-style-transfer-model	-	-	-	-
Affine-few-shot-style-transfer-model	39.89	45.34	19.13	23.49

Table 6.2: Results from style-transfer DA techniques. Unsupervised-style-transfer-model is pre-trained on synthetic images, in other words; weight-initialized with Synthetic-model. It’s then fine-tuned on 500 synthesized hybrid images (with ground truths from SimSalma and styles from the real-world). The resulting model will be used as weight-initialization of Few-shot-style-transfer model, where it’s further fine-tuned on 19 real-world images. A last attempt is tried with weight-initializing with Unsupervised-style-transfer-model, then fine-tuned on 19 real-world images with affine data augmentation. All style-transfer techniques show a decline in performance, most likely due to synthesizer not designed for segmentation - and object detection problems. Few-shot-style-transfer diverge during fine-tuning, so no results is given here.

6.4 Synthetic data performance

Without much real-world training data, this experiment did not involve a descent control model trained on real-world data, which could be used as a ground basis when compared against domain adapted models. However, results can still be compared to the control-model *Pure-target-model*. This comparison shows that regular fine-tuning only gave a relatively minor boost on Real-world-mix (27.48 mAP(%) over 24.48 mAP(%)). However, seen in relation to tests on the synthetic domain, it is clear that Pure-target-model is massively overfitted to camera-angles, light conditions, and fish-orientations, showing that pre-training on synthetic images then fine-tuning on real images, maintains fish-general features. More tests on out-of-sample real-world images should be done, but this suggests that synthetic data would be better than Pure-target-model, at least. Furthermore, using data augmentation in fine-tuning can ramp up the performance to 41.85 mAP (%), replicating 75% of how well Synthetic-model did in the synthetic domain and achieving almost twice as much as the (overfitted) Pure-target-model on real-world-mix. This shows that at least in a scenario where real-world training data is scarce - synthetic data in CNNs can represent a significant game-changer on performance.

Chapter 7

Discussion

7.1 Potential sources of error

7.1.1 Sampling bias and covariance

In this experiment, we did not have a real-world test-set fully representing the real world without a strong relationship between camera angles on the training set and test set. This could have affected our results, making Real-world-mix and Real-world-sequential (cropped) appear better than they would if compared to out-of-sample camera angles. However, the fact that regular fine-tuning using 19 images, produced such good results without altering its performance on the SimSalma dataset, suggests that much of the performance gain was due to real DA, not just being biased towards fish orientations. Furthermore, fine tuning with affine augmenter would perform much better than Pure-target-model, suggesting that at least 16.88% of the mAP points can be attributed to the model being general (*Pure-target-model* vs *fine-tuning w/ affine aug.* in table 6.1) .

However, the fact that Real-world sequential (uncropped) did poorly on many DA methods, shows that at least some types of real images will have a hard time being detected when DA is performed using fine-tuning. However, we can't know for sure if Real-world-sequential (uncropped) did poorly because of sampling bias. That is, since it was all filmed from a single camera angle, in a span of two seconds, with fish being smaller on average from the training set, it could simply be a dataset which magnifies a single troublesome camera angle. It could also be that it shows an important trend representing the real world. Since we can't know for sure, we should only take Real-world-mix into consideration, which we argue is reliable. Furthermore, we can take additional test sets into consideration when they show a trend, which they all do, relative speaking.

7.2 Gradual fine-tuning

There are some observations with gradual fine-tuning that show conflicting suggestions:

1. In real-world domain, referring to Real-world-mix in table 6.1:

   (a) *Gradually fine-tuning (using 19 real images)* (no data augmentation) gives a **slight better** performance compared to its counterpart *Regular Fine-tuning (using 19 real images)* (28.51 mAP(%) vs 27.48 mAP(%) respectively)

   (b) *Gradually fine-tuning w/ affine data aug*, show **worse results** compared to its counterpart regular *fine-tuning w/ data aug.* (36.45 mAP (%) vs 41.85 mAP (%) respectively).

2. In synthetic domain, referring to SimSalma in table 6.1:

   (a) *Gradually fine-tuning (using 19 real images)* (no data augmentation) gives a **slight worse** performance compared to its counterpart regular *Regular Fine-tuning (using 19 real images)* (55.25 mAP(%) vs 56.83 mAP(%) respectively)

   (b) *Gradually fine-tuning w/ affine data aug*, show **better** results compared to its counterpart regular *fine-tuning w/ data aug.* (36.91 mAP(%) vs 24 mAP(%) respectively)

Given that the reduction on SimSalma in observation 2a) is so small, it seems like the method of gradual fine-tuning has the capability of maintaining domain-invariant features across both domains while tune-tuning. The large difference in observation 1b) is mysterious, but it could perhaps have something to do with the number of epochs spent on fine-tuning headers (860 epochs before overfitting became apparent). It could be that when only focusing on these layers, overfitting to the new target dataset breaks co-adapted interactions between feature extractor and overheads learned in the source. We can observe a rapid decline on the synthetic source domain in figure 6.4 very early on, suggesting losing domain invariant features in overheads. Hence, domain invariant co-adapted interactions will naturally diminish in significance. Additionally, the gradual decline on real-world-sequential (seen in dark blue in 6.4) could be attributed to it relying some on features discovered trained on the source, but as the new features discovered in headers backpropagate into the whole network, the last domain-invariant features get ”overwritten”. If such is the case, it could be that gradual fine-tuning is still a good approach, but opening up new layers must be done before the new fine-tune loss reaches a minimum in the current stage, as this would be too late. However, more research is needed on this.

Another theory is that gradual fine-tuning gives a win-lose trade while training for a long time. By noting observation 2b), we can see that gradual fine-tuning can have the capability of maintaining cross-domain performance, but at the same time, this produces a worse result in the target domain. It could be that non-gradual fine-tuning is better on the real-world target domain (observation 1b) due to it overfitting to the real-world, while gradual fine-tuning is held back due to the method forcing it to stay cross-domain invariant. However, since this is only significantly the case using affine augmentation, we can’t know for sure if this theory is true or not.

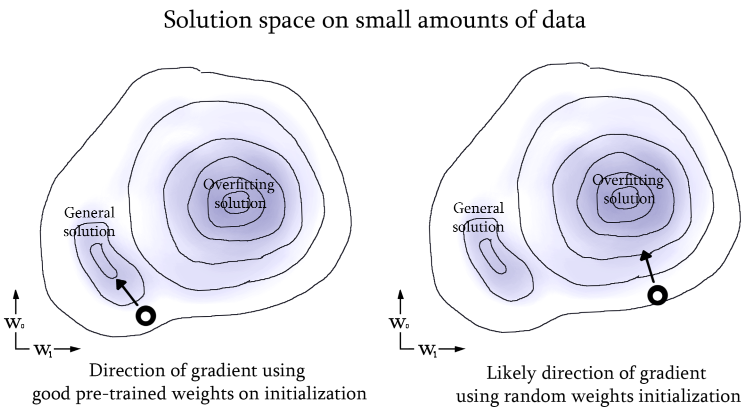
7.3 Nature of synthetic data

7.3.1 Covariate shift magnitude

Taking what we have learned from this experiment, combining it with the results from the project thesis [22] and section 3.2.2 shows that dataset bias plays a strong role in pushing CNNs trained on computer graphics to perform poorly on realistic images. However, at the same time, some facts might indicate that the real covariate shift is in fact, quite small.

- Regular fine-tuning (no augmentation) with just 19 images on synthetic-model managed to increase the cross-domain score a threefold, suggesting that synthetic-model already was generalized well beyond the dataset, but it only needed a little push in the target direction.
- Regular fine-tuning did not reduce the score on the source (synthetic) domain, showing that overfitting to sea conditions, fish orientation, and size, did not occur, and suggesting that the fine-tuned model is so general that domain invariant features have been learned.
- Fine-tuning on 19 images would usually result in massive overfitting (this we could see from the first epoch on fine-tuning Supervised-style-transfer-model), but since this didn't happen during fine-tuning of synthetic-model before the 23rd epoch, this suggests that something interesting was going on. Having such room for improvement, using gradients produced from such a small label subset, must mean that the direction of the mean gradient for all weights over these 19 images, was actually pointing in the direction of the true target domain local minimum, or close to it. This would likely not happen if weights were initialized independently to the source domain, suggesting that the general solution for the source and target domain is close. See figure 7.1 below, for an elaboration of this principle.
- In the project thesis [22], only 10 images did the same job.

Having a small covariate shift may or may not be common for other problems than computer graphics vs real world, but considering that few-shot domain adaptation is a well-established problem [5] [48] [49], it is unlikely that this holds true for most domain adaption scenarios. Is the covariate shift in computer graphics vs real world smaller than most scenarios? This we can't say for sure, but at least it seems to be small. Another question which remains; if the general solution in target domain is close to the general solution in the source domain, why is source model (synthetic-model) performing so bad without DA, tested on real pictures from the target domain? A possible answer might be that minor differences in activation inside the network accumulate and result in a "chaotic behavior" at the output. Though this is speculation, one fact might support this claim. Considering that Mask-RCNNs complicated pipeline which uses many concepts - if one thing goes wrong, the deeper layers will stop performing. For instance, even though the Mask header performs well if the RPN is poorly trained, mask predictions will be absent. Thus, a small push in the right direction for RPN will activate mask header performance. During fine-tune training jobs, an observation was made that the RPN often started to overfit before the mask header, supporting this claim.



Solution space on small amounts of data
General solution
Overfitting solution
W₀
W₁
Direction of gradient using good pre-trained weights on initialization
General solution
Overfitting solution
W₀
W₁
Likely direction of gradient using random weights initialization

Figure 7.1: The image show solution space in relation to weights for a neural network, here in two dimensions (2 weights). When there is a small amount of data in our fine-tuning training sample, the solution space usually prone towards overfitting. In this solution space, the general solution is still a solution, but it would likely not become apparent that the solution is better (less costly in error), unless we add a lot of data which in effect would move the overfitted solution gradually towards the general solution. However, with only a small amount of data, if fine-tuning / training starts with weights unrelated to the optimal solution, the initial position would most likely be closer to overfitting, thus gradient would point towards overfitting. On the other hand, if weights are moving towards the cavity of the general solution during training, the probability of weights having started close to it is high, which points to source domain and target domain being strongly related.

Chapter 8

Conclusion and Future Work

8.1 Conclusion

In this master thesis, it has been shown that domain adaptation can be achieved using synthetic data from computer graphics, with advanced architectures such as Mask-RCNN. It has also been shown that classical methods are useful when doing domain adaptation. Furthermore, we can extend many CNNs for visual applications with advanced DA methods which can be preferable if possible, but it is hard to implement in practice. To sum up our findings, this section will conclude our findings points wise.

8.1.1 On synthetic data and domain adaptation

- Based on the literature reviewed in section 3.2.2 and the performance on the real-world domain from our model trained on synthetic images of salmon, we can safely say that computer graphics often introduce a covariate shift in CNNs which substantially hinders the performance in-the-wild.

- Even though CNN models trained on synthetic data often come with a covariate shift, they can hold general features which are hidden within layers, and these can be accessed by using various domain adaptation techniques. Our experiment suggests that CNNs trained on realistic computer graphics should have a small covariate shift between the real world, and because of this, the shift can substantially be reduced by using simple few-shot or semi-supervised fine tuning. This makes synthetic data by computer graphics preferable over labeling whole datasets of real-world images when used with CNNs.

- Even though the covariate shift between synthetic data and real images might be small, it seems to result in a very large reduction on performance in-the-wild, at least for Mask-RCNN. A theory was put forward in the discussion-chapter, that this behavior is due to Mask-RCNNs complex sub-networks stacked on top of each other which adds layers of complexity on each other. We can predict that similar large, complex CNNs will behave poorly in-the-wild, but that fine-tuning can reduce this.

- Further improvements can be probably be done to reduce target domain loss using more advanced DA methods such as discrepancy methods, adversarial discriminate and generative methods, autoencoder reconstruction methods and style-transfer methods, referring to the end of literature study, section 4.7.1 for an overview of these. However, most of these methods are designed in the context of classification. If advanced DA methods were to be used on an arbitrary CNN architecture, it requires extending the architecture, which can be challenging if the inner workings of the architecture are unknown to the programmer, thus the workload should be weighted up against classical methods. Also, one needs to be creative in order to extend them for other visual applications. Some possible suggestions on extensions were also presented in the section on survey discussion in the literature survey (see section 4.7).

- More advanced DA methods exist in the literature which we didn't have time to cover, but the ones mentioned in section 4.7.1 are the most extensively studied and used and represents principles which are common across the literature. The scientific community has lately begun to look into proposing such extensions beyond classification [44].

- Due to time constraint, we didn't manage to compare how well regular fine-tuning did against more advanced DA techniques, but as a method within itself, it has much potential. Moving from a computer graphics domain towards the real, with fine-tuning we should not be surprised if we get a 50% performance gain, even with a small amount of fine-tuning data. With the correct type of data augmentation, 75% can be achieved. With this fact in mind, that such good performances can be achieved using just classical DA methods, it is conceivable that combining these with more advanced DA methods might make synthetic data, combined with DA, a good alternative to labeling real data.

8.1.2 More on Fine-Tuning

- Since fine-tuning is universal on all end-to-end CNNs, they represent a good alternative to advanced DA methods when the inner workings of the CNN architecture are complicated and difficult to extend, or if sufficient amount of target data is available.

- Combining fine-tuning with data augmentation should always be considered to gain a boost on performance and reduce the chance of overfitting, but the type of data augmentation must be carefully selected, as our experiment showed, that different types of data augmentation results in very different performance. Furthermore, with too much data augmentation, the adapted model might start getting a large dataset bias towards higher concept within the fine-tuning training set. To avoid this, one must make sure that the few labeled examples augmented are good representatives for the whole target dataset.

- Fine-tuning too much on target data might result in loss of cross-domain performance. If a CNN needs to perform well in multiple domains, it is recommended to monitor the loss of source domain during fine-tuning, as well as the target.

- Due to time constraints in our experiment, we didn't do fully gradually fine-tuning, that is; we didn't gradually open up all layers, as we only did this in three stages. However, the results in this thesis suggest that gradually opening up layers can give a slight performance boost, but this didn't hold when using data augmentation. The cause for this is not known. It could be that overtraining on each stage reduce co-adapted interactions between neurons, such that one should try to stop very early - opening up new layers, to counteract this. Keeping an eye on source domain loss as well as target domain loss might give a good indication on when to stop and move on to the next layer.

8.1.3 On style-transfer

Based on findings by Dundar[21], Csurka [34] and Bousmalis [35] (see section 4.3.3), style-transfer represents state-of-the-art methods for DA in CNN visual applications and should be considered as it allows for unsupervised training of a CNN without knowledge of the inner architecture of the preferred CNN. However, in the context of segmentation and object detection, or any application where objects need to have unique features associated with them; the type of network used for synthesizing training examples must be designed for such applications. The style-transfer-network should not introduce noise conflicting with the main task of the network, as most likely became the problem during the experiment put forward in this master thesis.

8.2 Future work

We have been given a small glimpse into the full picture of DA and how it relates to computer graphics. If research could continue, some objectives could further emphasize findings in the master thesis and/or add useful knowledge to the field.

8.2.1 Replication

The same experiment as put forward in this thesis should be replicated with more real-world test data. Since the experiment put forward contained some training set biases not guaranteed to be absent from the test set, the test data should also represent various camera-angles and light-conditions from the real world. Additionally, a model only trained on the real-world domain, such as Pure-target-model, can act as a better control base if real-world training images are plenty. Furthermore, gradually opening up layers for fine-tuning should be tested with more stages than just three to see if trends can be replicated. The time spent on each stage might also reveal an important variable affecting DA performance. Also, getting experimental support for the workings of our proposed advanced Mask-RCNN DA architectures in section 4.7.4 could represent an interesting comparison between classical methods and more advanced techniques.

8.2.2 Computer graphics

We identified a possible close relationship between the real and computer graphics in weight-space, but it would be interesting to identify the behavior of this relationship as realism in computer graphics drops towards less realism. Will the covariate shift drop linearly, exponentially and at what stage would the graphics be considered less useful for DA? An experiment can be proposed, where semi-supervised fine-tuning with real images is done on various Synthetic-models, were each model represent a ”step” in graphics adding to the realism. For all of these steps, we can see how performance after fine-tuning decline as a function of ”realism”. Such information would be important in order to know how much time should be spent on modeling synthetic data.

8.2.3 Advanced DA

In this master thesis, implementing advanced DA methods was down prioritized. As mention in the literature study, how computer-generated images will perform with such DA techniques, is not well studied. For a future work, it is recommended to implement many of the classification architectures mentioned in this thesis, such as DANN, BSW, JAN, DDC, ADDA, DRCN and Style-transfer, and compare their performance on computer-generated images to see which method works best. Even though instance segmentation became the focus of our study, classification has come a long way, making such comparison studies possible. Furthermore, this master thesis did not prioritize how domain adaptation can be done in Recurrent neural networks and video with time series. A literature study on such cases would be very interesting. Lastly, as many DA methods in classification and object detection were left out, more literature could be reviewed.

Bibliography

[1] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.

[2] Karl R. Weiss, Taghi M. Khoshgoftaar, and Dingding Wang. A survey of transfer learning. *Journal of Big Data*, 3:1–40, 2016.

[3] Quarterly Review of Biology. Oxford Dctionaries. `https://en.oxforddictionaries.com/definition/overfitting`, 1930. [Online; accessed 24-July-2018].

[4] Gabriela Csurka. Domain adaptation for visual applications: A comprehensive survey. *CoRR*, abs/1702.05374, 2017.

[5] Saeid Motiian, Quinn Jones, Seyed Mehdi Iranmanesh, and Gianfranco Doretto. Few-shot adversarial domain adaptation. *CoRR*, abs/1711.02536, 2017.

[6] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *CoRR*, abs/1802.03601, 2018.

[7] Shaoqing Ren, Kaiming He, Ross B. Girshick, and Jian Sun. Faster R-CNN: towards real-time object detection with region proposal networks. *CoRR*, abs/1506.01497, 2015.

[8] Ross B. Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *CoRR*, abs/1311.2524, 2013.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[10] Ross B. Girshick. Fast R-CNN. *CoRR*, abs/1504.08083, 2015.

[11] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[12] Unknown. Image aquired from webpage with url. `https://blog.csdn.net/u013010889`, 2018. [Online; accessed 24-August-2018].

[13] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. *CoRR*, abs/1608.02192, 2016.

[14] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. *CoRR*, abs/1705.05065, 2017.

[15] Open Source Robotics Foundation. Gazebo. `http://gazebosim.org/`, 2018. [Online; accessed 20-January-2018].

[16] Nvidia. Nvidia isaac sdk - accelerate your creation of autonomous machines. `https://developer.nvidia.com/isaac-sdk`, 2018. [Online; accessed 24-August-2018].

[17] Kiran Varanasi Kripasindhu Sarkar and Didier Stricker. Trained 3d models for cnn based object recognition. *Proceedings of the 12th International Joint Conference on Computer Vision*, pages 130–137, 01 2017.

[18] Matan Sela Elad Richardson and Ron Kimmel. 3d face reconstruction by learning from synthetic data. *3D Vision (3DV), 2016 Fourth International Conference on*, 2016.

[19] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. *Proceedings of 30th IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[20] Param S. Rajpura, Ravi S. Hegde, and Hristo Bojinov. Object detection using deep cnns trained on synthetic images. *CoRR*, abs/1706.06782, 2017.

[21] Aysegul Dundar, Ming-Yu Liu, Ting-Chun Wang, John Zedlewski, and Jan Kautz. Domain stylization: A strong, simple baseline for synthetic to real image domain adaptation. *CoRR*, abs/1807.09384, 2018.

[22] Magnus Reiersen. Synthetic data by 3d computer graphics in convolutional neural networks. *NTNU*, 2017.

[23] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[24] Judy Hoffman. Office dataset: Domain Adaptation Project . `https://people.eecs.berkeley.edu/~jhoffman/domainadapt/`, 2018. [Online; accessed 25-February-2018].

[25] Yann LeCun. MNIST handwritten dataset. `http://yann.lecun.com/exdb/mnist/`, 2018. [Online; accessed 25-February-2018].

[26] Artem Rozantsev, Mathieu Salzmann, and Pascal Fua. Beyond sharing weights for deep domain adaptation. *CoRR*, abs/1603.06432, 2016.

[27] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: an astounding baseline for recognition. *CoRR*, abs/1403.6382, 2014.

[28] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? *CoRR*, abs/1411.1792, 2014.

[29] Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. Best practices for fine-tuning visual classifiers to new domains. In *ECCV Workshops*, 2016.

[30] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.

[31] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

[32] Fujun Luan, Sylvain Paris, Eli Shechtman, and Kavita Bala. Deep photo style transfer. *CoRR*, abs/1703.07511, 2017.

[33] Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer. *CoRR*, abs/1701.01036, 2017.

[34] G. Csurka, F. Baradel, B. Chidlovskii, and S. Clinchant. Discrepancy-based networks for unsupervised domain adaptation: A comparative study. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 2630–2636, Oct 2017.

[35] Konstantinos Bousmalis, Nathan Silberman, David Dohan, Dumitru Erhan, and Dilip Krishnan. Unsupervised pixel-level domain adaptation with generative adversarial networks. *CoRR*, abs/1612.05424, 2016.

[36] Yaroslav Ganin and Victor Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1180–1189. JMLR.org, 2015.

[37] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. Adversarial discriminative domain adaptation. *CoRR*, abs/1702.05464, 2017.

[38] Eric Tzeng, Judy Hoffman, Ning Zhang, Kate Saenko, and Trevor Darrell. Deep domain confusion: Maximizing for domain invariance. *CoRR*, abs/1412.3474, 2014.

[39] Baochen Sun and Kate Saenko. Deep CORAL: correlation alignment for deep domain adaptation. *CoRR*, abs/1607.01719, 2016.

[40] Mingsheng Long, Jianmin Wang, and Michael I. Jordan. Deep transfer learning with joint adaptation networks. *CoRR*, abs/1605.06636, 2016.

[41] Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. Deep reconstruction-classification networks for unsupervised domain adaptation. *CoRR*, abs/1607.03516, 2016.

[42] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. Domain separation networks. *CoRR*, abs/1608.06019, 2016.

[43] Sicheng Zhao, Bichen Wu, Joseph Gonzalez, Sanjit A. Seshia, and Kurt Keutzer. Unsupervised domain adaptation: from simulation engine to the realworld. *CoRR*, abs/1803.09180, 2018.

[44] Yuhua Chen, Wen Li, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Domain adaptive faster R-CNN for object detection in the wild. *CoRR*, abs/1803.03243, 2018.

[45] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott E. Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: single shot multibox detector. *CoRR*, abs/1512.02325, 2015.

[46] Matterport. Mask R-CNN for Object Detection and Segmentation. `https://github.com/matterport/Mask_RCNN`, 2018. [Online; accessed 25-February-2018].

[47] Kongsberg digital. Ocean Farming 1. TEKNA Havbrukskonferanse 2017 , 18.-19.September, Trondheim, 2018. [Online; accessed 25-July-2018].

[48] Akisato Kimura, Zoubin Ghahramani, Koh Takeuchi, Tomoharu Iwata, and Naonori Ueda. Few-shot learning of neural networks from scratch by pseudo example optimization. 2018.

[49] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. *CoRR*, abs/1804.09458, 2018.

[50] Christopher K. I. Williams John Winn Mark Everingham, Luc Van Gool and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88, 6 2010.

[51] Alexander Jung. Imgaug: This python library helps you with augmenting images for your machine learning projects. `https://github.com/aleju/imgaug`, 2018. [Online; accessed 25-February-2018].

[52] Cameron Cysmith. neural-style-tf. `https://github.com/cysmith/neural-style-tf`, 2018. [Online; accessed 24-August-2018].

[53] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[54] Derek Hoiem, Yodsawalai Chodpathumwan, and Qieyun Dai. Diagnosing error in object detectors. 7574 LNCS(PART 3):340–353, 2012.

Appendix

8.3 ResNet

ResNet will be used in our implementation of Mask-RCNN. ResNet is short for Deep Residual Network and was introduced by a paper from Microsoft Research in 2015, working close to the same team behind Mask-RCNN [53]. A ResNet is a way to structure deep learning layers such that we are able to stack more layers on each other without risking running into the problem of exploding/vanishing gradients. In theory, having a deeper and deeper network should improve the performance of the network as it would be able to learn more and more complicated concepts. However, in reality, stacking layers can accumulate derivatives during backpropagation thus the network weights will become more and more sensitive to minor changes the deeper you go. This problem can be overcome by regularization, correct weight initialization, and a good activation function, however for very deep networks it can still become a problem. The Residual Network makes a huge improvement to this by introducing the residual block, a type of layer which receives input from the previous layer, but also from two layers before the previous. This is basically a shortcut. Stacking such layers on top of each other can make the network very deep, up to hundreds of layers. The reason it works so well is a shortcut in the residual block which allows for a smooth passing of concepts learned early in the stack, such that the deepest layers don't "scramble" information. Testing out ResNet with different depths on the CIFAR-10 dataset, the team found that the mAP topped at around a hundred layers. Going to a thousand decreased the performance. In this thesis ResNet-101 (a Residua Network with 101 layers) will be tested on Mask-RCNN as the backbone feature extractor.

8.4 Formulas

8.4.1 Confidence Threshold

In the problem of image *classification*, solvers based on CNN-architectures often output predictions with distributions over all classes, where each output neuron represents a confidence that the given input belongs to the output-class. Similarly, in the problem of *object detection*, solvers based on CNNs often output bounding-boxes with a confidence that the bounding-box and class is correct. Here, the natural problem which occur is; at what threshold should be considered a valid detection? Using high thresholds will make the CNN make confident and accurate predictions, while at the same time, many half-good predictions are left out, which could be correct. Also, using a low threshold the CNN will make a lot of inaccurate predictions yet few ground truth objects will be left out. In a *pr*, all thresholds are kept to keep maximum information on a detector.

8.4.2 Intersection of Union

In the problem of object detection, predicted bounding boxes rarely intersect with objects perfectly. There is a need to validate what is considered a ”correct match”. Here, we need a measurement which is versatile enough to fit bounding boxes of all sizes, and still being allowed to vary by some offset. A common way is using the official PASCAL VOC [50] definition, *Intersection of Union* or UiO. Here, a prediction is considered a match if the intersection of two rectangles divided by the union is bigger than 0.5:

$$TP(Pr) = if \;\; \frac{Pr \cap Gt}{Pr \cup Gt} > 0.5 \tag{8.1}$$

where $TP(Pr)$ is True Positive given a prediction Pr and a ground truth Gt. For problems dealing with segmentation, using rectangles to get the area would not be correct. Instead, we would need to count up all the pixels within the mask-polygon since the shape is arbitrary. Intersection would be the amount of pixels which is shared, located at the same place, between two polygons.

8.4.3 Precision

By all the predictions outputted from a solver which is classified as a detection, the percentage of these predictions which is correctly classified is defined to be the *precision*. Mathematically it is defined like this:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{n} \tag{8.2}$$

where TP is the count of all predictions being labeled a detection (True Positive), FP is the count of all predictions wrongly labeled as a detection (False Positive), and n is the count of all predictions classified to be a detection. Precision should be calculated separately for each class in the test set. Also, the mathematical operation of dividing TP by n should be done after counting predictions on all images in the dataset [50]

8.4.4 Recall

Recall is defined as the fraction of ground truth instances correctly classified by the solver. Mathematically it is defined in the following matter:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{m} \tag{8.3}$$

where TP is the count of all predictions being labeled a detection (True Positive), FN is the count of all ground truth instances missed by the solver (False Negative) and m is the count of all ground truth instances. Like precision, recall should be calculated separately for each class in the test set. Also, the mathematical operation of dividing TP by m should be done after counting predictions and ground truths on all images in the dataset [50].

8.4.5 PR curve

The PR-curve is a recall - against - precision plot, varied at different thresholds. Here, all thresholds are being considered, from 0 to 1. Each threshold will produce a corresponding average precision and average recall. The mathematical definition is:

$$PRcurve_j(T) = (recall^T, precision^T) \tag{8.4}$$

where T is different threshold values, $recall^T$ is the average recall validate at this threshold, and likewise. $precision^T$ is the average precision validated at this threshold.

8.4.6 Mean average precision

Mean average precision or *mAP* for short, is defined as the area under the PR-curve between 0 and 1. The mathematical definition is:

$$mAP = \int_0^1 PRcurve(r)dr \tag{8.5}$$

Mean average precision is well used in information retrieval systems and in the object detection problems [50]. The area under the curve, is however, difficult to calculate. The reason for this, is that no ”true” curve is present. Using points, the area can only be estimated. Different techniques is used for estimating mAP, and the subject is much debated [54] [50]. The PASCAL COCO estimation use the so called 11-point interpolated mean average precision:

$$\sum_{i=0}^{k} p\frac{1}{k} \qquad \text{where} \qquad p = \max\left\{precision(r) \subseteq P | r \leq \frac{i}{k}\right\} \tag{8.6}$$

where $k = 10$, r is average recall values at different thresholds-values, and P is the set of all precision values at all thresholds. However, in this thesis we are comparing models which relative scores can vary in tiny differences based on what DA method we use; therefore we set k to 100 as it will give a more precise area under the curve.

8.5 Extra images: Synthetic-model performance

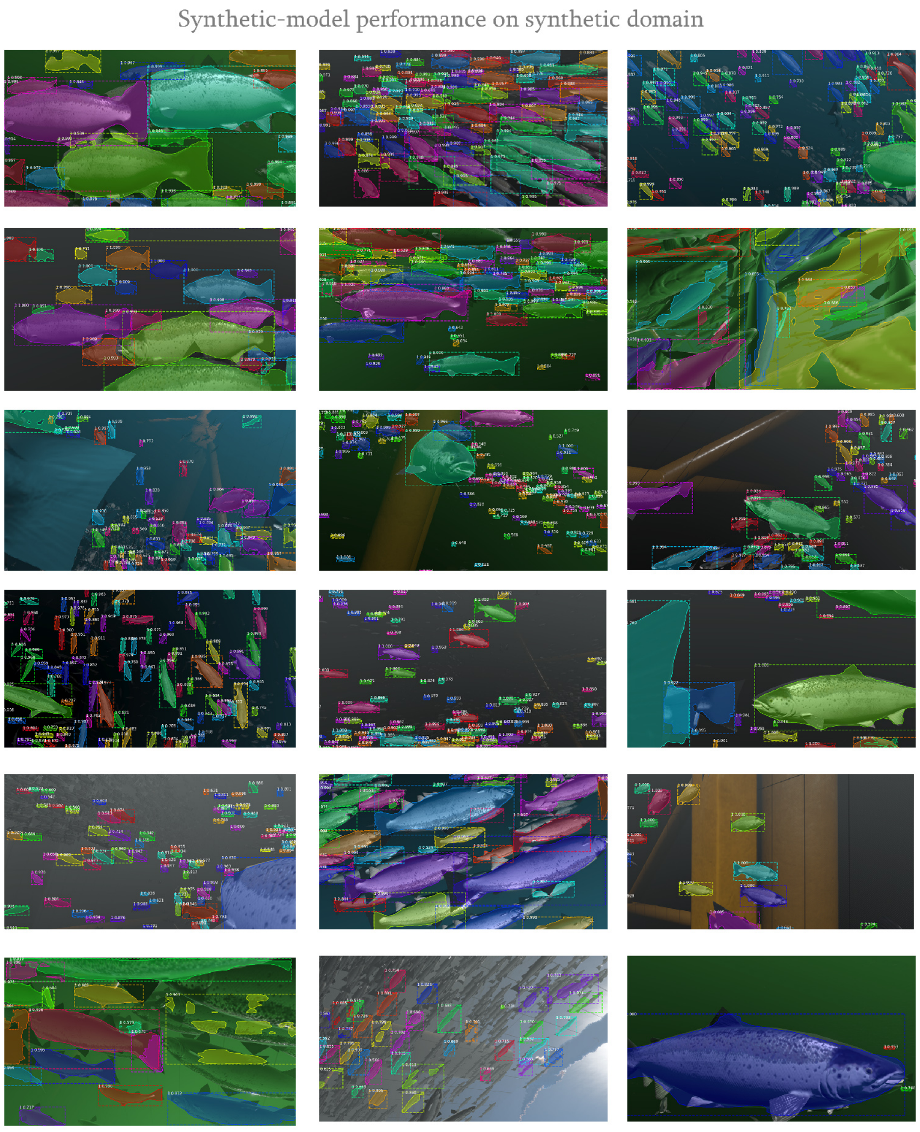


Figure 8.1: Experiment results: Pre-trained model (Synthetic-model) performance on its synthetic home-domain.

8.6 Extra images: Fine-tuned w/ affine aug. performance

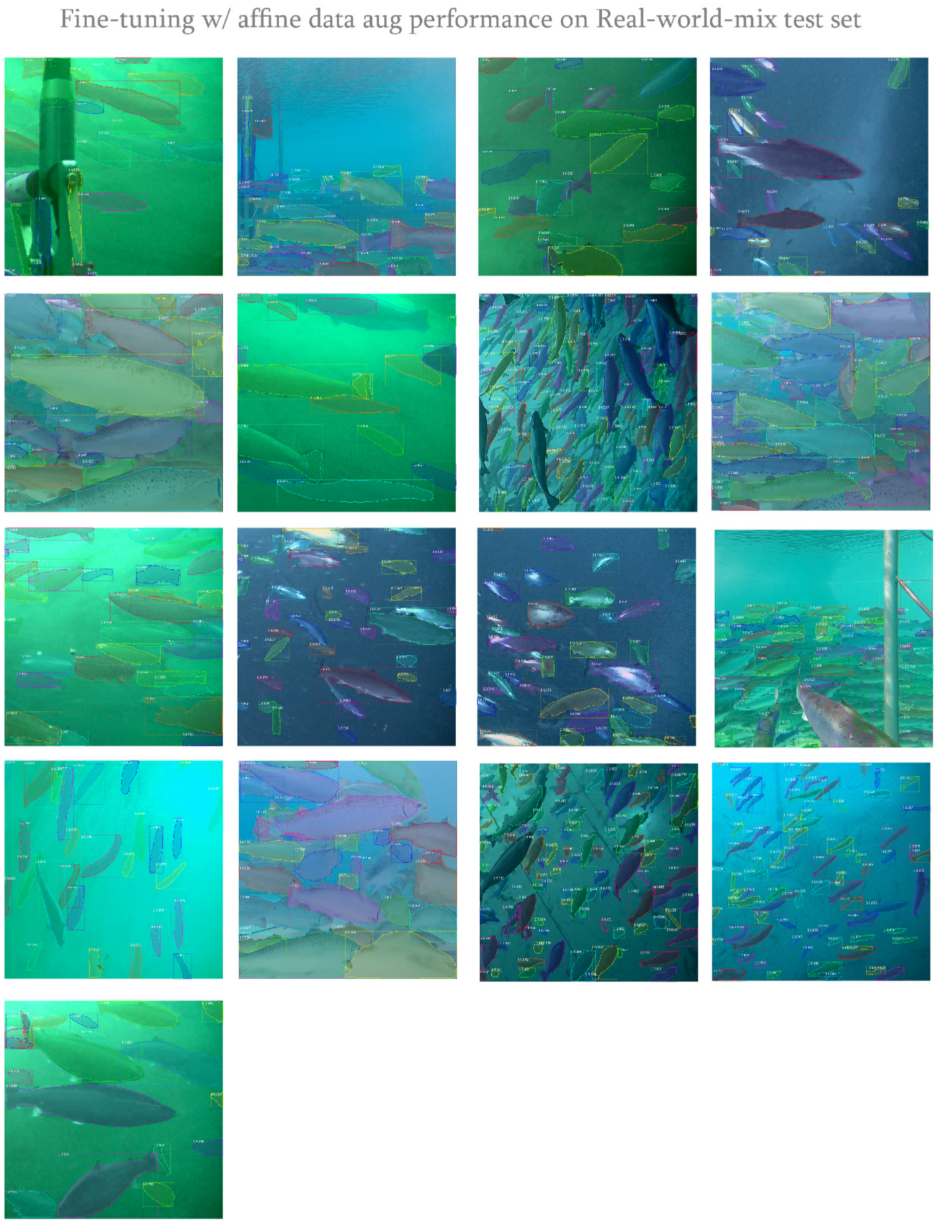

Fine-tuning w/ affine data aug performance on Real-world-mix test set

Figure 8.2: Experiment results: Visual demonstration of *Fine-tuning w/ affine data aug* performance on Real-world-mix test set. All images from the whole Real-world-mix test set is present in this image. *Fine-tuning w/ affine data aug* was the leading DA method from our results