



Technische Universität München

Department of Mathematics



Documentation

Semi-Supervised Labeling of Data with Varying Distributions

Robert Müller, Moritz Spielvogel, Xiaolu Xu

We assure the single handed composition of this documentation only supported by declared resources.

Garching,

Abstract

The purpose of the project is to find a work around method to alleviate the observable decrease in performance of our classification network when applied to data gathered from the customer (data usually sampled from different environments, e.g lighting, time). In this report, we propose an image translation network which can reuse the network trained on the original data. Moreover, this image translation can be applied to all future incoming customer data without compromising the performance to a degree.

To accomplish such a goal, we have first designed a ResNet model to simplify the original classification network, then built transfer learning models, lastly we modified the regularisation term to help the loss converge. The emphasis of the report is the current known relevant methods of building an image translation network, mostly different types of GANs. We have applied the most promising models on the data provided by the PreciBake company, and closely compared data before and after image translation network.

After performing experiments with different GAN models, it is hard to say that any model is superior to others. It is safe to assert that each model has its strength in specific role, and the accuracy depends on many other factors combined (e.g augmentation method). One finding of our investigation into the topic is that the data augmentation step can greatly bias our final accuracy. It is well justified to invest time on finding a good augmentation method. As the network tends to overfit the training data. Regularisation term may also play a role when tuning the network.

To summarize, our work mainly looks at GAN structured image translation networks. Currently, all GAN networks improve the performance of the classification network, but still are not able to raise the accuracy on the customer data to the level of the original data. To further improve this, we may need to have well collected data-sets, and insightful data augmentation methods.

Contents

1	Introduction	1
2	Motivation	2
2.1	PreciBake’s Challenge - Our Problem	2
2.2	Data and Task	2
2.3	Transfer Learning	4
2.4	Solution approach	5
3	Background	6
3.1	Classification	6
3.1.1	Introduction	6
3.1.2	ResNet	6
3.1.3	Augmentation	7
3.2	Image Translation	8
3.2.1	Introduction	8
3.2.2	Generative Adversarial Networks (GAN)	8
3.2.3	Wasserstein Generative Adversarial Network (WGAN)	9
3.2.4	Cycle-Consistent Adversarial Networks (cycleGAN)	11
3.2.5	Geometry-Consistent Generative Adversarial Networks (GcGAN)	12
3.2.6	Distance constraint Generative Adversarial Networks (distanceGAN)	14
3.2.7	U-Net	15

3.2.8	UNIT Model	16
3.3	Using GANs for Classification	18
3.4	Adversarial Training	20
3.4.1	Motivation	20
3.4.2	Unlabeled Data - Virtual Adversarial Training (VAT)	21
3.4.3	Loss Function	21
4	Experiments	23
4.1	Augmentation	23
4.2	Choice of Hyperparameters	28
5	Results	34
6	Conclusion	35
6.1	What has been learned?	36
6.2	Future Work	36

1 Introduction

Deep neural networks have recently driven the progress in various classification tasks for supervised and organised data. Supervised learning means that we have data consisting of samples and respective labels. Unfortunately, we find often that if we put the well-trained models into use in real world, the performance is worse than expected because the data from real world are messy and unorganised. Furthermore we encounter novel conditions which never occurred in the training process.

If we observe data it is per se unlabeled. With the increase in amount and capabilities of sensors there is a huge increase in the amount of unlabeled data. Thus, one approach would be to try to label all unlabeled data manually to come back to the supervised case. If we decide however to use the unlabeled data directly we speak from unsupervised learning. A scenario that occurs often in practice is the coexistence of labeled and unlabeled data, which we call semi supervised learning. It promises to make use of vast amounts of unlabeled data for our tasks.

As training a model for a new task from scratch takes a long time we would always like to exploit previous knowledge. Luckily, this works great for a lot of tasks. For example the geometric shapes bounding objects are similar across multiple domains. It might be even the case that the pre training data domain and our actual domain have a latent relationship and differ for example only in their lightning. In this essay, we focus on the ability of generalizing a model to conditions, which are different from the ones encountered during training, referred as transfer learning. And with the techniques based on transfer learning [14], we try to solve the problem, which encounters in real world and industry.

2 Motivation

2.1 PreciBake's Challenge - Our Problem

PreciBake is an Artificial Intelligence (AI) and sensor technology company providing software and hardware solutions to the professional baking industry. To achieve this goal, PreciBake came up with the concept of a virtual baker which functions as AI-empowered assistance system for ovens. It simplifies usability, processes and amplifies quality at the Point of Sale (POS) and bakery. After a successful learning phase, the Virtual Baker recognizes the product in the oven, automatically selects the baking program, and adjusts it independently as needed. It recognizes the actual amount of products in the oven and adjusts the baking program accordingly. [1]

However, sometimes the models were trained with organized and collected data have difficulties in recognizing sorts of different bread in real world due to the changes of light condition, camera angles, shapes of dough and so on. It is consuming both time and efforts if we collect data from new conditions, have to label it and finally retrain the model. As such we would like to constantly improve the model by constantly acquired unlabeled data. Specifically we look for methods that help to adapt pretrained models based on unlabeled data from a divergent situation.

2.2 Data and Task

For our Datalab project we are given data from two different domains of data which are the labeled PreciBake labor data called "precibake_data" or "domain Y" and unlabeled data of the "real" world or "fieldtest_data" resp. "domain X". Each domain has ten different classes of baking products: "Apfelecke", "Baguette", "Baguettebrötchen", "Buttercroissant", "Kaiserbrötchen", "Laugenbreze", "Laugenstange", "Laugenzopf", "Pain au Chocolate" and "Tomatenstrudel". Our task is to build a model which is able to classify the unlabeled samples in the fieldtest domain.

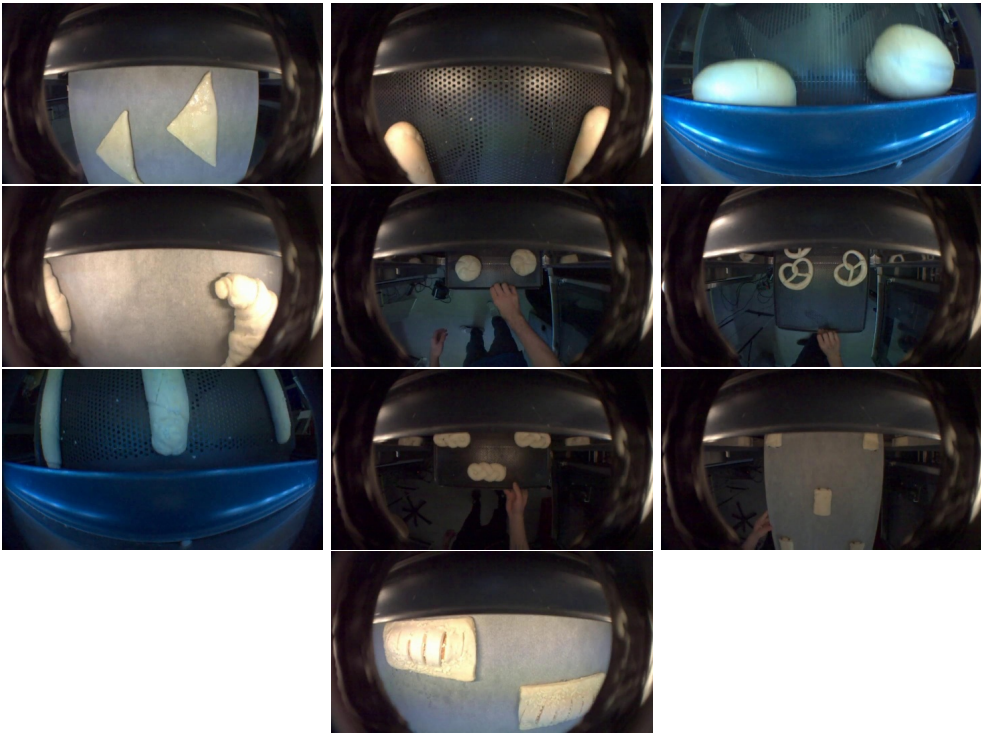


Figure 1: One image of each class of precibake_data

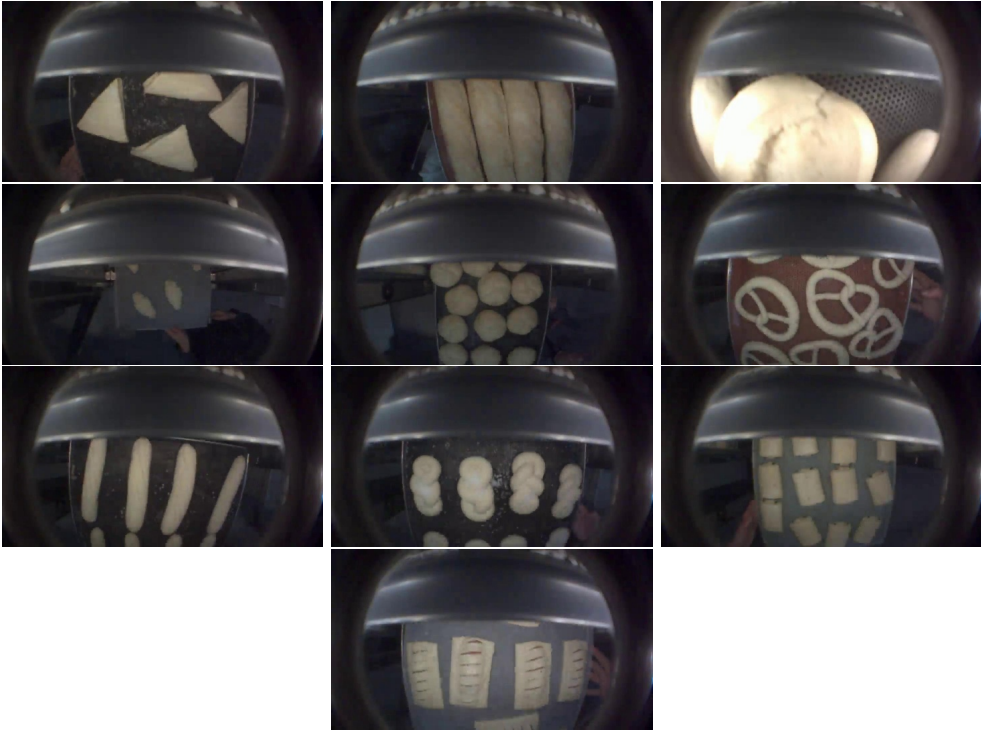


Figure 2: One image of each class of fieldtest_data

2.3 Transfer Learning

The classical supervised learning scenario of machine learning, if we intend to train a model for some task and domain Y, we assume that we are provided with labeled data for the same task and domain. Consequently, our model is trained to solve the problem in this specific domain. Shown in figure 3:

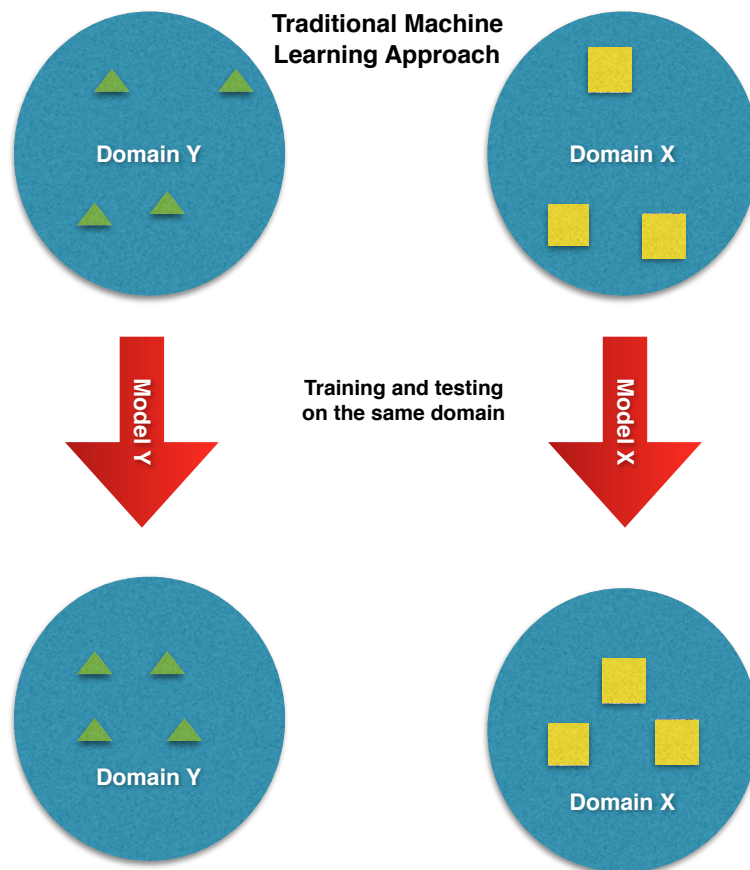


Figure 3: Classical ML

Transfer learning allows us to deal with these scenarios by leveraging the already existing labelled data of some related task or domain. With the model solving the source task in the source domain, the idea of applying it to our problem of interest as can be seen in figure 4.

Practically, the more knowledge we are able to transfer from source setting to our target task or domain, the better the methods which we used are.

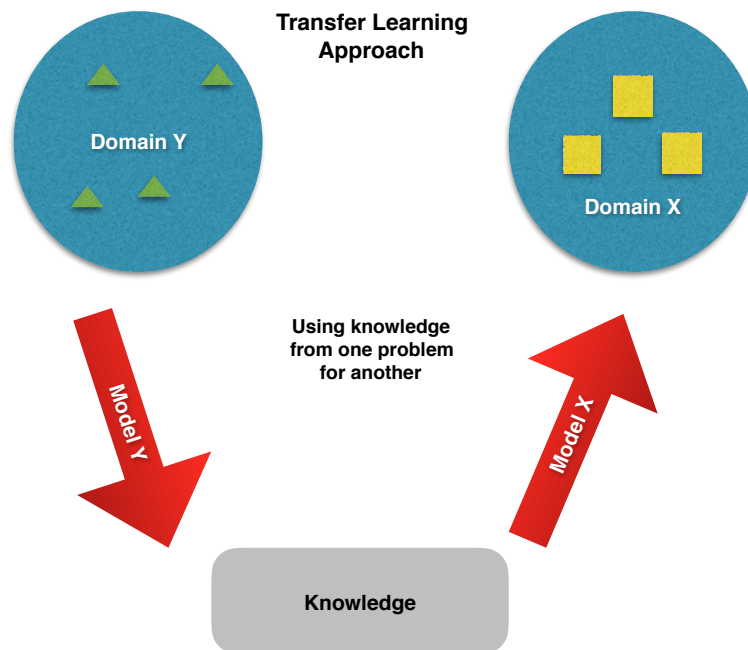


Figure 4: Transfer Learning

2.4 Solution approach

Our Semi-Supervised Learning Approach is the following: We first used the labelled data to train a classifier. Afterwards we trained a transfer model with our unlabeled data and the labelled data to find a mapping from the unlabeled domain to the labeled one. At the end we used the mapping to transfer an image to the other domain and then we classified it using the classifier model. For the classification task we trained a ResNet18 [10] with the training data of our precibake dataset and tested it on the test data of the same domain.

The transfer models were trained with the training data of both domains (precibake and fieldtest data). They are Generative Adversarial Network (GAN) based and try to project the images of the domain X into domain Y .

Afterwards, we applied the transfer models on the test images of fieldtest data to transfer them to the domain of precibake data.

Then, we tested the transfer models using our ResNet18 model to see if they lay now in the distribution of precibake data. We also tried to augment the precibake data (as can be seen in 3.1.3) to get more variance in the distribution such that it might get close to the distribution of the other domain. Besides, we applied Virtual Adversarial Training, which provides a new loss function and generalizes the model to make it more adaptable.

3 Background

3.1 Classification

3.1.1 Introduction

The goal of the project is to perform image classification on photos taken under variate light condition. For classifying the data that are labelled and sampled form the source domain, the ResNet18 model is used. Deeper networks are difficult to train, therefore we follow the idea provided in this paper [10] to reformulate the layers. Furthermore, to fasten the model training, we utilize the ResNet pre-trained model by fixing the first few feature extraction layers and adapt the deep layers based on our custom data.

3.1.2 ResNet

The ResNet helps improving our model performance and avoid the problems brought by adopting deep neural networks. One observation is that deep neural network may have degrading performance compared to that of shallow network, even when more layers usually means more capacity of the network. In the deeper network, the additional layers may not better approximates the mapping than its shallower counter part, but rather act like an identity function which directly map input to output. The residual block is therefore introduced to make deep neural networks perform at least as good as shallow networks can. In a traditional plain block,

$$y = F(x)$$

where F stands for stacked neural network layers.

In a residual block,we optimise instead the function:

$$y = F(x) + x$$

The above equation can easily represent the identity function by setting $F(x) = 0$.

Among all the models currently available, we choose Resnet for its efficiency.

ResNet Structure The training part consists of three parts: data-loading, training, and tuning. Due to the characteristics of our training samples, lighting conditions are highly weighted when classifying the object, which is not desired. We want to build a model use mostly the shape and contour to distinguish different type of bread. This feature of the data set motivates us to carefully choose the augmentation method, since the augmentation methods can affect the final accuracy significantly.

In the data-loading part, instead of the standard data augmentation methods, we additionally use a pytorch project named Augmentor [4] to adjust color and lighting conditions in order to generate more training samples from the data-sets. The details of the augmentations will be explained in the following subsection.

In the training part, we do standard Adam batch gradient descent with Adam and minimise the mean squared error . Both the learning rate the the size of the epochs are treated as the hyper-parameters to be decided in the tuning part.

3.1.3 Augmentation

As only a few training samle of domain X are available, Augmentation is essential to gain more robustness of our model. Besides, those images are from the precibake lab and made under specific light conditions and camera angels. Therefore, it does not cover all possible conditions. But we need our model to be invariant under different light conditions and angels. Thus, will motivate us to include rotations and color transformations in the augmentations. Moreover, the classes have different amounts of images. This yields to an unequal weight of each class on the accuracy of our ResNet18 model. Thus, we will have an equal amount of images in each class after augmentation.

We will see in the section Experiments which augmentations we considered to use and which ones we actually used.

3.2 Image Translation

3.2.1 Introduction

As explained in "Methods", we need to have for the Semisupervised-Learning Task besides a classifier model also a transfer model which shifts the data to the domain on which the classifier is trained.

We have two different approach groups which we tested. The first group (cycleGAN like approaches) aims to learn a mapping directly from domain X to domain Y whereas the second group (UNIT model) aims to map from domain X to a latent space Z and then to domain Y .

3.2.2 Generative Adversarial Networks (GAN)

GANs [6] have recently achieved impressive results on a wide range of tasks, from text2image, image editing, representation learning to time series prediction. We focus on the use in domain adaptation. The basic idea behind GANs is to learn a generative models based on game theory. In the most general case we train a generator network $G(z, \theta)$ that takes in noise vectors z and aims to reproduce samples from the true distribution $p_{data}(x)$. The opponent of the generator is the discriminator network $D(x)$. In the course of training the discriminator is trained to distinguish samples from the real distribution $p_{data}(x)$ from the samples generated by the discriminator \pm . The generator is conversely trained to fool the discriminator. D and G play the two player minimax game:

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{p_z(z)} [\log (1 - D(G(z, \theta)))] \quad (1)$$

In our setting we consider however mappings between two domains X (data from the laboratory) and Y (data from the field test). We can draw samples from X and Y according to the marginals P_X and P_Y . As we want to perform semi supervised learning in the laboratory space X is suffices to consider a one sided mapping. Thus we need to learn a Generator G_{YX} s.t. $G_{YX}(Y)$ has the same distribution as X , i.e. $P_{G_{YX}(Y)} = P_x$. In addition we learn a discriminator D_X hat aims to distinguish real samples $x \sim P_X$ from translated samples $G_{YX}(y)$. Thus our GAN objective can be rewritten as:

$$\mathcal{L}_{gan}(G_{YX}, D_X, X, Y) = \mathbb{E}_{x \sim P_x} [\log D_X(x)] + \mathbb{E}_{y \sim P_Y} [1 - \log (D_X(G_{YX}(y)))] \quad (2)$$

In recent years a broad variety of GAN models emerged. In the subsequent chapters we introduce several advanced concepts such as cycleGANs.

3.2.3 Wasserstein Generative Adversarial Network (WGAN)

Let $G(z)$ be a generator that generates for any random vector $z \sim \mathbb{P}_z$ generates samples $\{G(z)\}$ that follow the resultig distribution \mathbb{P}_G . WGANs [2] aim to learn a generator network $G(z)$ that minimizes the Wasserstein distance between the distribution of generated samples \mathbb{P}_G and the real data distribution \mathbb{P}_r , i.e. we seek $\min_G W(\mathbb{P}_r, \mathbb{P}_G)$. The Wasserstein distance, also refered to as earth movers distance, is defined as:

$$W(\mathbb{P}_r, \mathbb{P}_G) = \min_{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_G)} \mathbb{E}_{(x,y) \sim \gamma} [\|x - y\|], \quad (3)$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_G)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are respectively \mathbb{P}_r and \mathbb{P}_G . Imagine \mathbb{P}_r and \mathbb{P}_G as two piles of earth. One can think intuitively of $\gamma(x, y)$ as indicating how much earth must be transported from x to y in order to transfer the one earth pile \mathbb{P}_r into the other earth pile \mathbb{P}_G . The Wasserstein distance can be intuitively grasped as the "cost" of the optimal transport plan. Unfortunately it is intractable to compute the minimum in 3. Using the Kantorovich-Rubinstein duality [21] we can however reformulate the Wasserstein distance:

$$W(\mathbb{P}_r, \mathbb{P}_G) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r} f(x) - \mathbb{E}_{x \sim \mathbb{P}_\theta} f(x), \quad (4)$$

where we take the supremum over all 1-Lipschitz functions $f : X \rightarrow \mathbb{R}$ and θ are the parameter of the generator G , in our case the weights of the neural network. Let $d(\cdot)$ be the l_2 metric in the input space. Thus the generator G produces samples $G(z, \theta) \sim \mathbb{P}_\theta$. We say a function $f : X \rightarrow Y$ is Lipschitz continuous, if there exists a real constant $K > 0$ s.t. for all $x_1, x_2, \in X$:

$$d(f(x_1), f(x_2)) \leq M \cdot d(x_1, x_2). \quad (5)$$

If instead of 1-Lipschitz functions we consider K -Lipschitz functions we end up with $K \cdot W(\mathbb{P}_r, \mathbb{P}_G)$. If we assume that all members of the function family $\{f_w\}_{w \in \mathcal{W}}$ are K -Lipschitz for some K we could solve:

$$W(\mathbb{P}_r, \mathbb{P}_G) = \max_{w \in \mathcal{W}} \mathbb{E}_{x \sim \mathbb{P}_r} f_w(x) - \mathbb{E}_{x \sim \mathbb{P}_\theta} f_w(x). \quad (6)$$

Assuming that the supremum in 4 is attained this yields a calculation of $W(\mathbb{P}_r, \mathbb{P}_G)$ up to a multiplicative constant. This leads us to the value function of the WGAN:

$$\min_G \max_{D \in \mathcal{D}} \mathbb{E}_{x \sim \mathbb{P}_r} D(x) - \mathbb{E}_{z \sim \mathbb{P}_z} D(G(z)), \quad (7)$$

where \mathcal{D} is the set of 1-Lipschitz functions. Here we take the previous function f as discriminator D . We refer for a more detailed derivation to the paper [2]. It is worth to mention that the 1-lipschitz constraint on f is enforced by clipping maximal weight values, i.e. the weigths of the discriminator are kept within a certain range $(-c, c)$, where c is a hyperparameter. As a result of the enforced 1-Lipschitz constraint probelms suc as poor image quality or non convergence might arise, when c is not tuned correctly. This issues have been adressed by suggested improvement of the WGAN. One notably improvement is the gradient penalty (GP) method calles WGAN-GP, which exploits the fact that a differentiable function f is 1-Lipschitz if and only if its gradients have norm at most 1 everywhere. Instead of weight clipping WGAN-GP penalizes the discriminator model if the gradient norm moves away from the target norm value 1:

$$\mathcal{L} = \mathbb{E}_{\hat{x} \sim \mathbb{P}_G} D(\hat{x}) - \mathbb{E}_{x \sim \mathbb{P}_r} D(x) + \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} \left[(\|\nabla_{\hat{x}} D(\hat{x})\| - 1)^2 \right], \quad (8)$$

where we sample $\hat{x} = t\tilde{x} + (1-t)x$ with $t \in [0, 1]$, i.e. along the line between both samples.

The WGAN-GP has however weaknesses as well. Unlike the weight clipping te gradient penalty can not penalize everywhere within a finite number of training iterations. This means the GP term is only applied to \hat{x} which we sampled, which might leave significant parts of the support domain uninvestigated. [22] illustrate this at the example of the observed data point which are distributed accoring to \mathbb{P}_r and follow a underlying manifold. At the beginning of the training stage we find however that \mathbb{P}_G can very distant from the true data manifold, thus \hat{x} being far apart from the true distribution. Thus the Lipschitz continuity over the manifold of \mathbb{P}_r is not until the generator G becomes sufficiently good. [22] propose to solve this by additionally laying a Lipschitz continuity condition over the manifold of the real data $x \sim \mathbb{P}_r$. In order to penalize violations of the Lipschitz continuity 5 they propose the folling soft consistency term (CT):

$$CT|_{x_1, x_2} = \mathbb{E}_{x_1, x_2} \left[\max \left(0, \frac{d(D(x_1), D(x_2))}{d(x_1, x_2)} - M' \right) \right]. \quad (9)$$

arguments from the paper [22]

Thus they propose the consistency regulation:

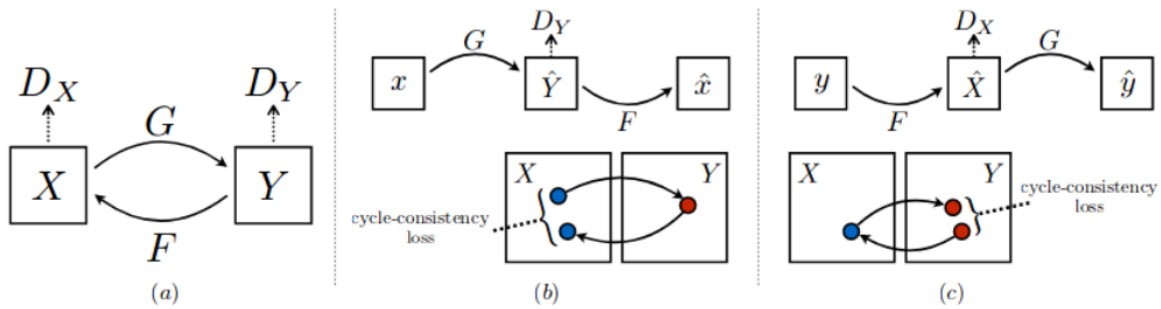
$$CT|_{x_1, x_2} = \mathbb{E}_{x \sim \mathbb{P}_r} \left[\max \left(0, d(D(x'), D(x'')) \right) + 0.1 \left(0, d(D_-(x'), D_-(x'')) \right) \right]. \quad (10)$$

Combing this with the objective function for the discriminator of WGAN-GP we get the new objective function:

$$\mathcal{L} = \mathbb{E}_{\tilde{x} \sim \mathbb{P}_G} D(\tilde{x}) - \mathbb{E}_{x \sim \mathbb{P}_r} D(x) + \lambda_1 \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}} \left[(\|\nabla_{\hat{x}} D(\hat{x})\| - 1)^2 \right] + \lambda_2 CT|_{x_1, x_2}. \quad (11)$$

3.2.4 Cycle-Consistent Adversarial Networks (cycleGAN)

CycleGANs were introduced by [24] for problems where the goal is to learn a mapping between a source domain X and a target domain Y . Our goal is to train the generator such that it generates images $\mathbb{P}_G(x)$ that are indistinguishable from the samples from domain Y according to \mathbb{P}_Y . We use a architecture as displayed in (a) in ???. This means that in addition to the generator $G : X \rightarrow Y$ we have a generator $F : Y \rightarrow X$. In both domains we have a discriminator D_X respectively D_Y that tries to distinguish samples from the domain from fake samples. The idea behind cycleGANs is that we wish for $F(G(x)) \approx x$ and vice versa $G(F(y)) \approx y$, i.e. if we map a sample to the other domain and map it subsequently back we want to be close to our initial sample. This is illustrated in picture (b) and (c) in ???. The cycleGAN loss consists of



Graph credit to [24]

Figure 5: cycle gan

three summands. Firstly we formalize the already motivated circularity constraint:

$$\mathcal{L}_{cycle}(G_{X,Y}, G_{Y,X}) = \mathbb{E}_{y \sim P_y} \|y - G_{X,Y}(G_{Y,X}(y))\|_1 + \mathbb{E}_{x \sim P_x} \|x - G_{Y,X}(G_{X,Y}(x))\|_1. \quad (12)$$

Similar to the normal GAN setting we have the adversarial loss \mathcal{L}_{GAN} from the discriminators D_X and D_Y aiming to distinguish true and fake samples. As a third term we consider the identity constraint. If we apply the generator G_{XY} to a sample from domain Y we want G_{XY} to perform an identity mapping. As we consider both directions we get:

$$\mathcal{L}_{ID}(G_{X,Y}, G_{Y,X}) = \mathbb{E}_{x \sim P_x} \|x - G_{Y,X}(x)\|_2 + \mathbb{E}_{y \sim P_y} \|y - G_{X,Y}(y)\|_2 \quad (13)$$

Thus we use the full objective:

$$\mathcal{L} = \mathcal{L}_{GAN} + \mathcal{L}_{cycle} + \mathcal{L}_{ID} \quad (14)$$

to train our generator. The discriminator is again trained to detect fake samples. The cycle consistency constraint is a form of regularisation and prevents as such over fitting. The cycleGAN structure with two generators and two discriminators as well as the cycle consistency term where used in many of our experimental architectures. The idea of having two domains and generators in between gives rise to a variety of methods with a similar circular constraint. Instead of the circular constraint they consider for example the distance between samples in both domains. The the subsequent chapters we give an overview over several methods.

3.2.5 Geometry-Consistent Generative Adversarial Networks (GcGAN)

Motivation Again the goal is to learn a mapping from domain X to domain Y , but GcGAN or Geometry-Consistent Generative Adversarial Networks (for One-Sided Unsupervised Domain Mapping)[5] has the advantage over cycleGAN that it does not need to learn a mapping from domain Y to domain X as illustrated in figure 4. The geometric idea behind GcGANs is that it should map an image in the same way as it would map a transformed image.

Explanation GcGAN uses a geometric transformation function for having a metric of the performance of the generators. This is done in the following way: First, a generator tries to generate from an image x of domain X an image y' of domain Y . Then, this image will be transformed with the geometric transformation function f to $f(y')$. Afterwards, the same image of domain X will be transformed using f to x and then be translated to domain Y using another generator. Finally, the transferred image \tilde{y}' will be transformed using the inverse function of f to $f^{-1}(\tilde{y}')$. Then we compare \tilde{y} with $f^{-1}(\tilde{y}')$ and $f(y')$ with \tilde{y}' . This comparison is described in mathematical terms in the paragraph "Loss functions" as \mathcal{L}_{geo} .

Geometric Transformation There are multiple definitions for geometric transformations. For our case it is necessary that a geometric transformation has an inverse function[15]. Since we need it to be able to determine \mathcal{L}_{geo} . Then it is important to classify those transformations because we want to choose our hyperparameters according to different classes of geometric transformations. For example one can classify them by the dimension of their operand sets or by the properties they preserve[23]. We will do the latter one as it is more useful in our application. Hence, we use the classes translation, rotation, affine transformation and perspective transformation for our choice of a geometric transformation function.

Loss function The loss function of the GcGAN model has multiple terms:

$$\begin{aligned} \mathcal{L}_{GcGAN} = \lambda_{geo} \mathcal{L}_{geo}(G_{XY}, G_{f(X)f(Y)}, X, Y) + \mathcal{L}_{gan}(G_{XY}, D_Y, X, Y) + \mathcal{L}_{gan}(G_{f(X)f(Y)}, D_{f(Y)}, X, Y) \\ + \mathcal{L}_{identity}(G_{XY}, X, Y) + \mathcal{L}_{identity}(G_{f(X)f(Y)}, X, Y) \end{aligned} \quad (15)$$

with

$$\begin{aligned} \mathcal{L}_{geo}(G_{XY}, G_{f(X)f(Y)}, X, Y) = \mathbb{E}_{x \sim P_X} (\|G_{XY}(x) - f^{-1}(G_{f(X)f(Y)}(f(x)))\|) \\ + \mathbb{E}_{x \sim P_X} (\|G_{f(X)f(Y)}(f(x)) - f(G_{XY}(x))\|), \end{aligned} \quad (16)$$

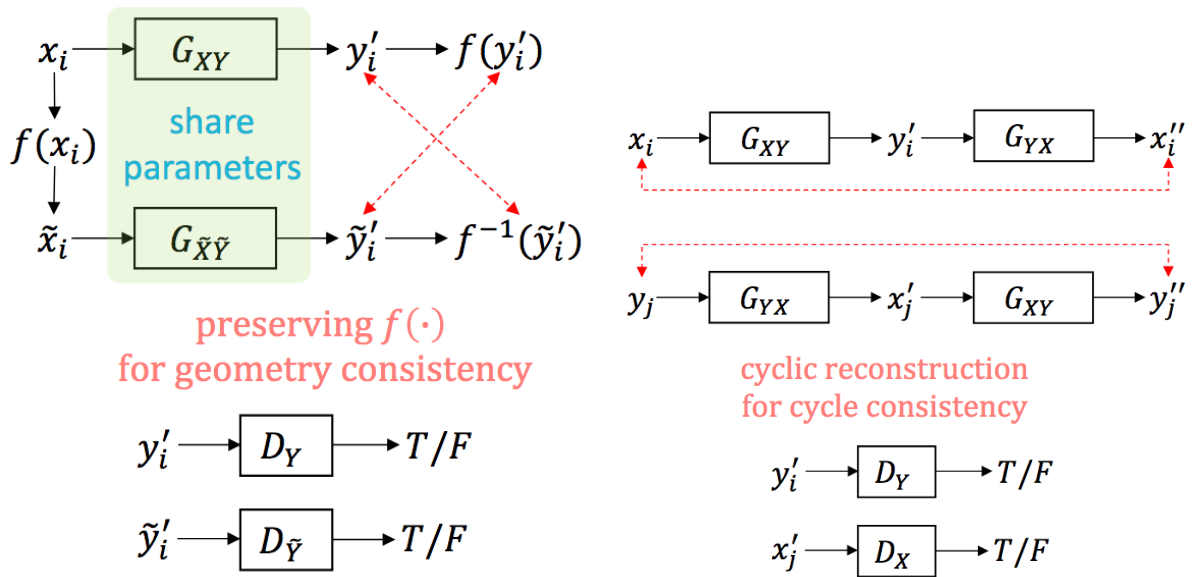
$$\mathcal{L}_{identity}(G_{XY}, X, Y) = \mathbb{E}_{y \sim P_Y} (\|G_{XY}(y) - y\|), \quad (17)$$

$\lambda > 0$ is a weight factor of the geometric loss.

\mathcal{L}_{geo} checks whether the is invariant under a geometric transformation f or not (as described above in more detail).

$\mathcal{L}_{identity}$ aims to ensure that the generator would generate the same picture y if it gets as an input a picture y from domain Y .

$\mathcal{L}_{gan}(G_{XY}, D_Y, X, Y)$ as described in the section GANs.



Graph credit to [5]

Figure 6: Illustration of the difference between GcGAN and CycleGAN[5]

3.2.6 Distance constraint Generative Adversarial Networks (distanceGAN)

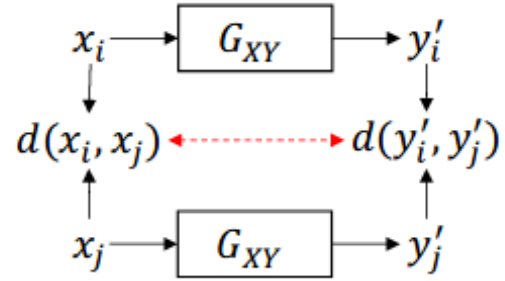
The geometric idea behind distance constraint GANs, outlined by [3], is that the distance between two samples y_i and y_j in domain Y should be approximately preserved after mapping to domain X , thus $d(y_i, y_j) \approx a \cdot d(G_{YX}(y_i), G_{YX}(y_j)) + b$, where $d(\cdot)$ yields the distance between two points, a is a linear coefficient and b the occurring bias. We denote by μ_X, μ_Y the means of distances of all possible pairs of samples $(x_i, x_j) \in X$ and $(y_i, y_j) \in Y$. Let σ_X, σ_Y denote similarly the standard deviation of this distances. Then we can define the pairwise distance consistency loss:

$$\mathcal{L}_{dis}^{pair}(G_{YX}, X, Y) = \mathbb{E}_{y_i, y_j \sim P_Y} [|\phi(y_i, y_j) - \psi(y_i, y_j)|] \quad (18)$$

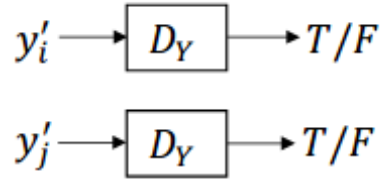
$$\phi(y_i, y_j) = \frac{1}{\sigma_Y} (\|y_i - y_j\|_1 - \mu_Y) \quad (19)$$

$$\psi(y_i, y_j) = \frac{1}{\sigma_X} (\|G_{YX}(y_i) - G_{YX}(y_j)\|_1 - \mu_X) \quad (20)$$

Instead of using pairwise distances and standard deviations we could apply the same logic to individual samples. This means specifically that we expect the difference between the left and right (or upper and lower half) of a picture to be the same in both domains. This approach has the advantage that the precomputation of the statistics is considerably faster than for the pairwise case as we consider each picture only once. Let for this $L, R : \mathbb{R}^{h \times w} \rightarrow \mathbb{R}^{h \times \frac{w}{2}}$. We denote by $\tilde{\mu}_X, \tilde{\mu}_Y$ the means of distances of the left and right (upper and lower) half of the picture, i.e. $\mu_C = \frac{1}{n} \sum_j \|L(x_j) - R(x_j)\|_1$ $x_i \in X$ and $y_i \in Y$. Let $\tilde{\sigma}_X, \tilde{\sigma}_Y$ denote similarly the standard deviation of this distances. We define the self distance consistency loss:



preserving $d(\cdot)$
for distance consistency



DistanceGAN

Graph credit to [5]

Figure 7: distance constraint gan

$$\mathcal{L}_{dis}^{self}(G_{YX}, X, Y) = \mathbb{E}_{y_i, y_j \sim P_y} [|\phi(y_i, y_j) - \psi(y_i, y_j)|] \quad (21)$$

$$\phi(y) = \frac{1}{\tilde{\sigma}_Y} (\|L(y) - R(y)\|_1 - \tilde{\mu}_Y) \quad (22)$$

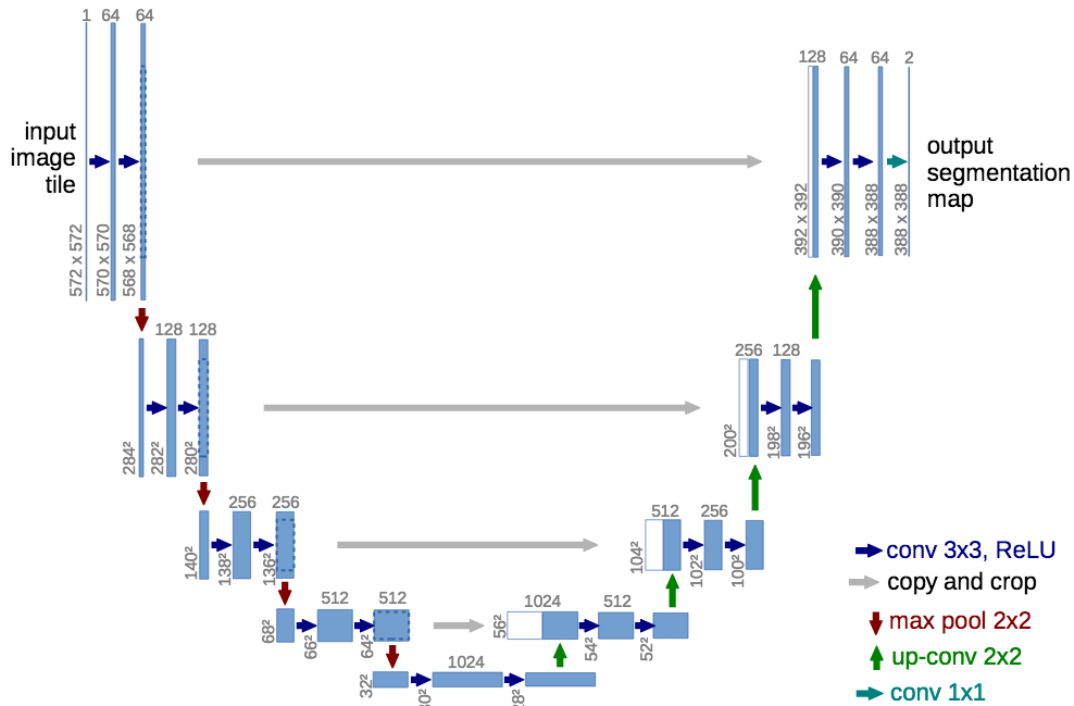
$$\psi(y) = \frac{1}{\tilde{\sigma}_X} (\|L(G_{YX}(y)) - R(G_{YX}(y))\|_1 - \tilde{\mu}_X) \quad (23)$$

In our implementation we perform precomputation for all the occurring statistics, i.e. $\mu_X, \mu_Y, \sigma_X, \sigma_Y$ of our data.

3.2.7 U-Net

For the generator of our Cycle GAN like approaches we used the U-Net [16] for several reasons. First of all, it does not have any fully connected layer. Therefore, we are able to use any size of an image. This helps to train our model in a much faster way as we can change the image size during training. This enables us to start with small (downscaled) images and get first results for the parameters. Afterwards, we refine them by bigger (less down-scaled) images. And at the end, we can use the original images. Besides, U-Net combines the information from the downsampling path with the information of the upsampling path (as can be seen in figure 3). Thus, it combines information from the location in the image with the context of this location. This helps the generator to segment the images well.

In comparison to the generator used in [17] U-Net is a way faster Generator.



Graph credit to [16]

Figure 8: U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. [16]

3.2.8 UNIT Model

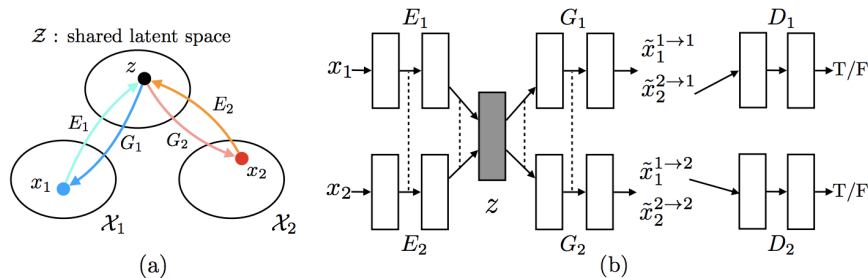
Introduction Unsupervised Image-to-Image Translation Network (UNIT) is introduced by Ming-Yu Liu, Thomas Breuel, Jan Kautz with codes and additional results available in this paper [12]. UNIT model aims to solve the problems where we have two groups of data taken under different domains, and we want to transform an image from one domain to another. The problem get complicated when we do not have one to one correspondence between pictures that are relevant. UNIT model tries to learn a joint distribution between two distribution. The UNIT framework is based on generative adversarial networks (GANs) section3.2.2 and variational auto-encoders (VAEs).

In the previous sections of this documentation, cycle GAN section3.2.4 and distance GAN section3.2.6 have been introduced and detailed explained. UNIT model bases on the idea of competing GANs, but also makes advanced assumption about the intrinsic structure of the data set. With the additional information, the choice of the generator can be narrowed down without downgrading the performance of our network.

The supporting assumption underlying the UNIT model is that both the training space (domain X) and testing space (domain Y) share one same latent space (named Z). We can consider the

latent space as the extracted essence of the object that stays invariant in either domain. In this view, pictures in both space X and Y are representations of the same object under different environments. To see this idea in real-life applications, the latent space can be the human with certain face features and the space A and B can then be different races or ages.

Figure 9: Illustration of the unit model with latent space assumption



Graph credit to [12]

In part(a): X_1, X_2 corresponds to training and the testing domain. Z is the latent space we assume to exist.

In part(b): E represents the auto-encoder, G is the generator, D has the goal to distinguish whether the given image is generated by the encoding from domain X_1 or X_2 .

The shared latent space assumption allow us to perform image to image translation, by first encode the input into the latent space and then apply the generator that are specific to the desired domain.

Framework of the UNIT Model The shared latent space assumption is implemented by the design of the weight sharing structure of the network in combined with the variational auto-encoders. The auto-encoder is performing the task of representing an given image by a one-dimensional array. This array intuitively count the relative weight of the each feature in this image. We use variational auto-encoder which tries to bring the distribution of these generated arrays to that of Gaussian random variable. This array can be treated as a representation of the image in the latent space Z . To ensure a smooth transition, the last few layers of the encoders E_1, E_2 and the first few layers of the encoders G_1, G_2 are fixed by sharing the weight of the network between both domain X and Y .

A side note that an implementation of weight-sharing structure does not guarantee Z produce latent encoding of semantic meanings. However, we can use adversarial training to produce latent space that connect the encoders and generators of these two domains.

Losses in the UNIT model If we follow the idea that there exists a latent space, it is inferred that the image translation must satisfy the Cycle-Consistency constraint. To be more specific, an image in domain X , should barely vary after being encoded to domain Z , translate to domain Y , and then transform back to domain X by reversing the process. Noticeably, cycle-consistency is also a constraint for cycle GAN in previous section.

In the unit model, the goal is to optimise 6 losses, 3 losses in each direction of transformation (space X to Y and space Y to X). The three losses in one direction is the the loss from the discriminator of the GAN, the KL divergence of the VAE and loss from the cycle-consistency.

Figure 10: Formula of the losses in UNIT model

$$\min_{E_1, E_2, G_1, G_2} \max_{D_1, D_2} \mathcal{L}_{\text{VAE}_1}(E_1, G_1) + \mathcal{L}_{\text{GAN}_1}(E_2, G_1, D_1) + \mathcal{L}_{\text{CC}_1}(E_1, G_1, E_2, G_2) \\ \mathcal{L}_{\text{VAE}_2}(E_2, G_2) + \mathcal{L}_{\text{GAN}_2}(E_1, G_2, D_2) + \mathcal{L}_{\text{CC}_2}(E_2, G_2, E_1, G_1). \quad (2)$$

Graph credit to [12]

3.3 Using GANs for Classification

The previously outlined GANs provide the ability to transform pictures from domain X to domain Y , where we perform classification with a pretrained model, i.e. a Resnet. A different approach is to use the GAN discriminator for classification. Our domain adaptation problem is a classification problem with K classes. We denote by l_i the class label of sample x_i , thus our labelled data have the form (x_i, l_i) . We begin by considering the standard classification problem with K classes, thus a sample x is classified into one of K . Therefore we train a model taking x as input and output of a K -dimensional vector of logits $[\ell_1, \dots, \ell_K]$. Applying the softmax function we derive the probabilities of x belonging to class j as:

$$p_{\text{model}}(l_i = j|x) = \frac{\exp(\ell_j)}{\sum_{i=1}^K \exp(\ell_i)} \quad (24)$$

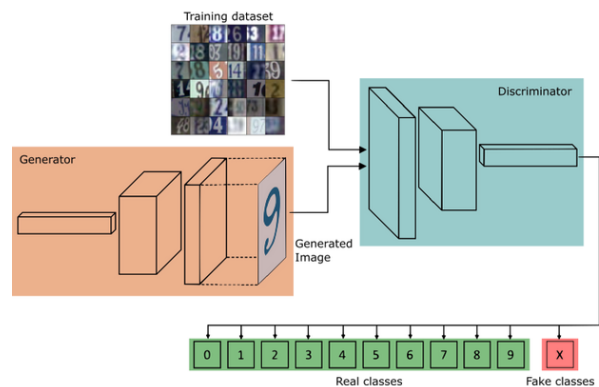
The model is trained by minimisation of the cross entropy between the label data and the model predictive distribution $p_{\text{model}}(l|x)$.

If we now consider a multi-domain classification problem we might want to use the discriminator of the GAN for classification. The standard architectural choice [18] [22] is to pick a discriminator with $(K + 1)$ outputs, where output j corresponds to the probability of class j and the $K + 1$ th output corresponds to the probability of being a fake samples produced by the generator, i.e. an sample that is detected as being mapped from the source domain. The resulting loss for the discriminator D can be decomposed into the loss for labelled samples (\mathcal{L}_l^D) and unlabelled samples (\mathcal{L}_{ul}^D), $\mathcal{L}^D = \mathcal{L}_l^D + \mathcal{L}_{ul}^D$. The supervised loss for our classifier is given as:

$$\mathcal{L}_l^D = -\mathbb{E}_{x, l \sim p_{\text{data}}(x, l)} \log p_{\text{model}}(l|x, l < K + 1) \quad (25)$$

and the unsupervised loss for the discriminator is given as:

$$\mathcal{L}_{ul}^D = -\mathbb{E}_{x \sim p_{\text{data}}(x)} \log [1 - p_{\text{model}}(l = K + 1|x)] + \mathbb{E}_{x \sim G} \log [p_{\text{model}}(l = K + 1|x)]. \quad (26)$$



Graph credit to

<https://towardsdatascience.com/semi-supervised-learning-with-gans-9f3cb128c5e>

Figure 11: using the discriminator for classification in a semi supervised learning setting

Thus we obtain the total loss of the classifier:

$$\mathcal{L}^D = -\mathbb{E}_{x \sim p_{data}(x)} \log [p_{model}(l|x)] - \mathbb{E}_{x \sim G} \log (p_{model}(l = K + 1|x)). \quad (27)$$

Here we decomposed the total cross entropy loss into the stand supervised loss \mathcal{L}_l^D and a unsupervised loss \mathcal{L}_{ul}^D that corresponds to the standard GAN value as can be seen from substituting $D(x) = 1 - p_{model}(l = K + 1|x)$:

$$\mathcal{L}_{ul}^D = -\mathbb{E}_{x \sim P_x} \log D(x) - \mathbb{E}_{z \sim noise} \log (1 - D(G(z))) \quad (28)$$

which results in our domain adaptation problem between domains X and Y with respective data distributions P_X and P_Y to:

$$\mathcal{L}_{ul}^D = -\mathbb{E}_{x \sim P_x} \log D(x) - \mathbb{E}_{y \sim P_y} \log (1 - D(G(y))). \quad (29)$$

To train the generator [18] recommend to use feature matching in the discriminator, thus the generator matches expected features of generated and real samples. Let D_r denote the r -th layer in the discriminator, the generator minimizes the expected feature loss:

$$\mathcal{L}^G = \|\mathbb{E}_{x \sim P_x} D_r(x) - \mathbb{E}_{y \sim P_y} D_r(G(y))\|_2^2 \quad (30)$$

Alternatives are to use the classic GAN generator update rule. For the special case of WGANs [22] suggest to use the penultimate layer in the update rule of the generator and the following objective function for the discriminator:

$$\mathcal{L}^D = -\mathbb{E}_{x \sim p_{data}(x)} \log [p_{model}(l|x)] - \mathbb{E}_{x \sim G} \log (p_{model}(l = K + 1|x)) + \lambda CT|_{x', x''}, \quad (31)$$

with $CT|_{x', x''}$ as defined above.

We conclude by the observation that our classifier with $K + 1$ logits as output is indeed over determined. This is due to the fact, that subtracting a general function $f(x)$ form each logit, i.e. $l_i(x) \leftarrow l_i(x) - f(x), \forall i \in \{1, \dots, K + 1\}$ does not change the output of the softmax distribution. Thus it is possible to fix any logit to zero, w.l.o.g. we set $l_{K+1}(x) = 0$. Then \mathcal{L}_{ul}^D is again similar to the supervised loss with K classes and the classifier $D(x)$ is given as:

$$D(x) = \frac{\sum_{k=1}^K \exp(\ell_k)(x)}{\sum_{k=1}^K \exp(\ell_k)(x) + 1}. \quad (32)$$

In order the use the discriminator in our precibake setting we apply again a cycleGAN idea that is outlined in picture 12.

Totally the generator is training with the same loss function as in the normal cycleGAN, thus with the weighted sum of self constraint, adversarial constraint and cycle distance constraint:

$$\mathcal{L}^G = \mathcal{L}_{GAN} + \mathcal{L}_{cycle} + \mathcal{L}_{ID} \quad (33)$$

and for the discriminator a version of 31.

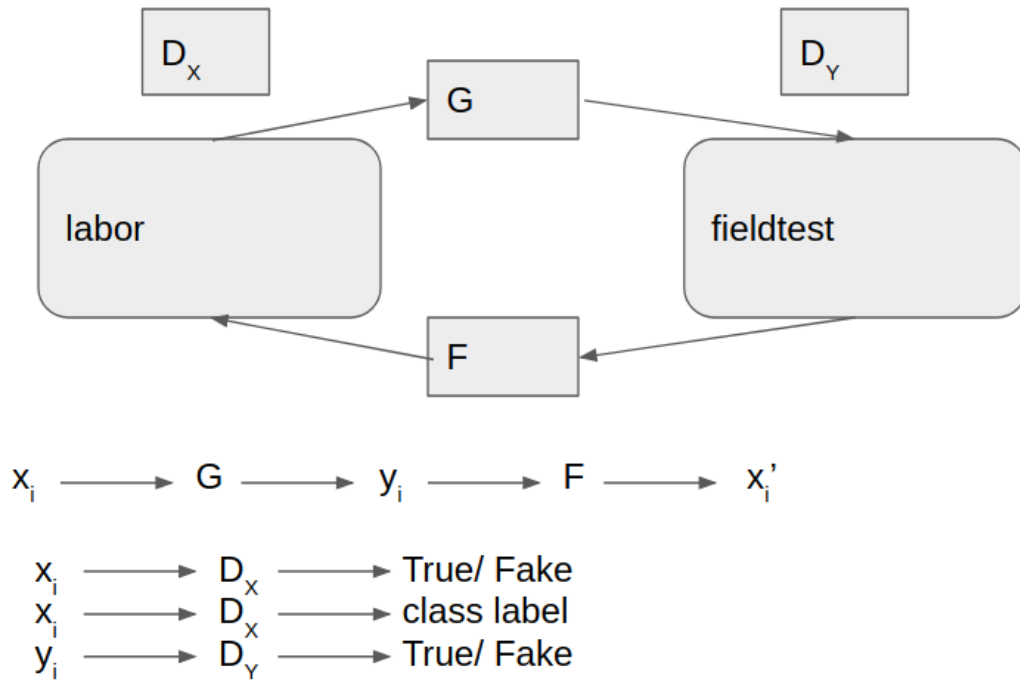


Figure 12: cycleGAN like architecture with 2 generators and discriminators that we use for classification. As we are only interested in classifying the samples in the labor domain it is sufficient to have a discriminator D_X with the ability to classify and a normal discriminator D_Y .

3.4 Adversarial Training

3.4.1 Motivation

[19] pointed out that neural networks learn input output mappings that are to a significant extent discontinuous. They show that it is possible to make a network misclassify an image by adding an for humans imperceptible perturbation. For a specific noise level $\epsilon > 0$ there is a perturbation deviating by less than ϵ from the original sample that maximises the networks prediction error. An example can be seen in picture 13.

Multiple methods have been proposed to compute the optimal perturbation. However most of this approaches rely upon the knowledge of the label of the prospective sample. For our task we face however a large amount of unlabeled data, that we also want to make use of. One method to do so is virtual adversarial training.

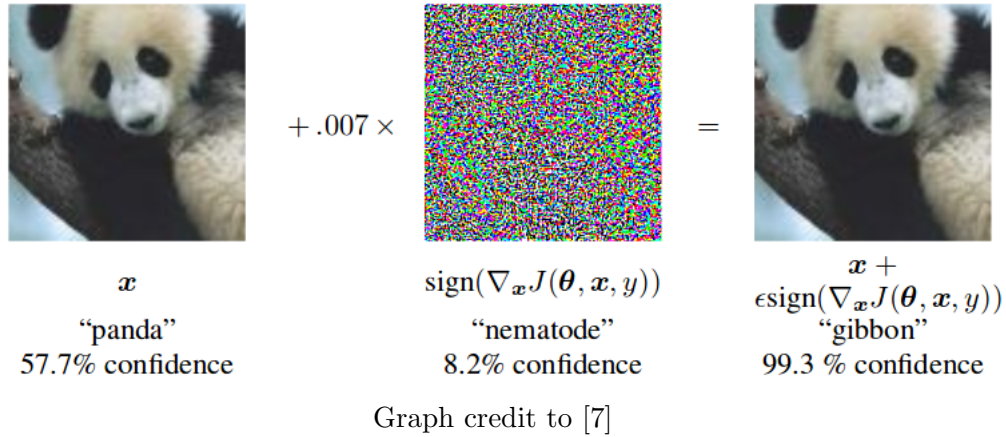


Figure 13: Example of adversarial training.

3.4.2 Unlabeled Data - Virtual Adversarial Training (VAT)

As we do not know correct labels for unlabeled data, [13] proposed a new regularisation technique which is a method that trains the output distribution to be isotropically smooth around each input data point by selectively smoothing the model in its most antiisotropic direction.

Virtual adversarial direction aims at the most antiisotropic direction. In contrast, adversarial direction introduced by Goodfellow et al [8] at an input data point is a direction of the perturbation that can most reduce the models probability of correct classification, or the direction that can most greatly deviate the prediction of the model from the correct label.

As showed in the figure 14 below , after several updates, the VAT can classify and generalise the model quite well.

3.4.3 Loss Function

To quantify the direction, VAT define the local distributional smoothness (LDS) to be the divergence- based distributional robustness of the model against virtual adversarial direction.

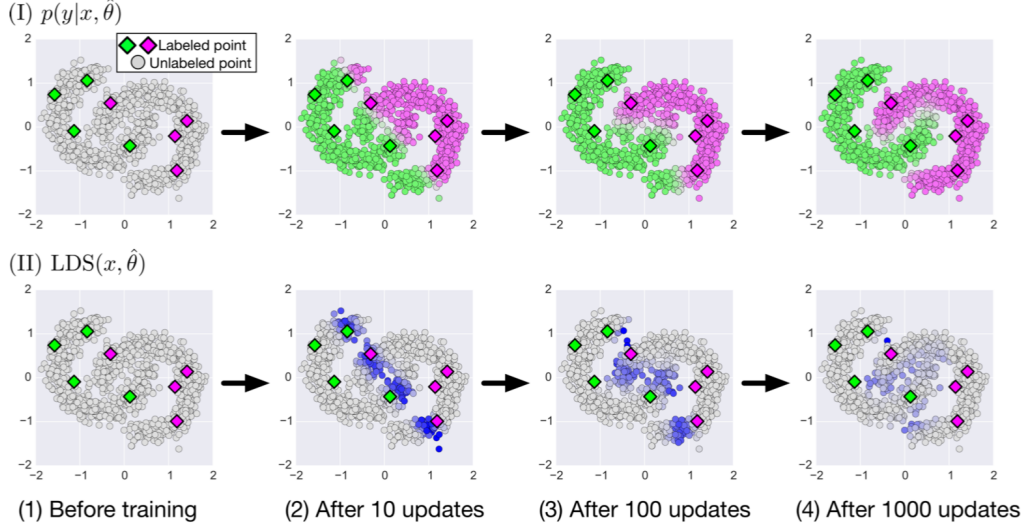
Let $D[p, p']$ be a non-negative function that measures the divergence between two distributions p and p' , for example, the cross entropy. And x_l is the input data, y is the output label .Then loss function of Adversarial Training is

$$L_{adv}(x_l, \theta) := D[q(y|x_l), p(y|x_l + r_{adv}, \theta)] \quad (34)$$

$$r_{adv} := \operatorname{argmax}_{r; \|r\| \leq \epsilon} D[q(y|x_l), p(y|x_l + r, \theta)] \quad (35)$$

Based on Adversarial Training ,let x_* represent either labeled data x_l or unlabeled data x_{ul} ,then the object function is now :

$$L_{adv}(x_*, \theta) := D[q(y|x_*), p(y|x_* + r_{qadv}, \theta)] \quad (36)$$



Graph credit to [13]

Figure 14: The local distributional smoothness (LDS)

$$r_{qadv} := \operatorname{argmax}_r; \|r\| \leq \epsilon D[q(y|x_*), p(y|x_* + r, \theta)] \quad (37)$$

Actually, there is no direct information about $q(y|x_*)$ and the current estimate $p(y|x, \hat{\theta})$ can be a great replace for it. Thus, we can get the virtual adversarial perturbation which represented by local distributional smoothness as below:

$$LDS(x_*, \theta) := D[p(y|x_*, \hat{\theta}), p(y|x_* + r_{qadv}, \theta)] \quad (38)$$

$$r_{qadv} := \operatorname{argmax}_r; \|r\| \leq \epsilon D[p(y|x_*, \hat{\theta}), p(y|x_* + r)] \quad (39)$$

The regularization term proposed is the average of $LDS(x_*, \theta)$ over all input data points:

$$R_{vadv}(D_l, D_{ul}, \theta) := \frac{1}{N_l + N_{ul}} \sum_{x_* \in D_l, D_{ul}} LDS(x_*, \theta) \quad (40)$$

Putting everything together [13] propose the following loss function,

$$l(D_l, \theta) + \alpha R_{vadv}(D_l, D_{ul}, \theta) \quad (41)$$

and it is obvious that VAT is a training method with the regularizer R_{vadv} . This methods is particularly useful as in reality there are orders of magnitude more unlabeled than labeled data, thus it is desirable to exploit them as well.

We use VAT to train the discriminator. Thus $l(D_l, \theta)$ denotes the loss function of the discriminator in the version as classifier of true and fake samples from the current domain or in addition as classifier of the respective kind of pastry.

4 Experiments

4.1 Augmentation

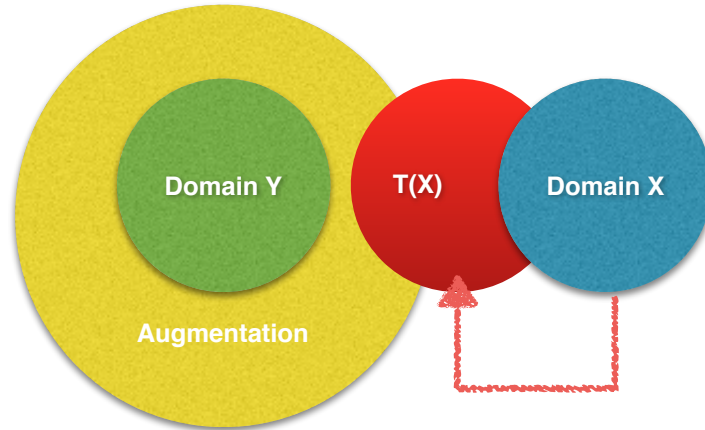


Figure 15: Illustration of the effect an Augementation might have on the data.

Domain Y is from the PreciBake lab and domain X from the customer’s site (later known as fieldtest). As one can see in figure 10, the effect of augmentation should be that it generates more variance such that the augmented domain Y would be closer to domain X. We also assume that our transfer models (in figure 10: $T(X)$) would shift domain X closer to domain Y. This would yield to the results that the ResNet18 model, which is trained on the augmented domain Y, would have a higher accuracy on the test data of domain X. We will see later for which transfer model and for which augmentations this will be the case.

Different Augementations In the following list are different augmentations presented. The accuracy during testing of each augmentation is displayed on the righthand side. We first used 7 different ways to edit an image, which were all ways we wanted to include in an augmentation. As the accuracy of this Augmentation was significantly low in comparison to train the ResNet18 on the non augmented data, we had to exclude some factors. After excluding only one factor which did not change the color and realizing that it does not lead to a higher validation accuracy, we started by using only one factor and to add one after the other to see which factors have an significant impact on the validation accuracy. Thereby, we can recognize that the brightness has an effect on the validation accuracy in "Augment6", "Augment8", "Augment9" and "Augment10". This yielded to a relatively high min- and max-factor for the brightness.

Table 1: All Augmentations, which we testet, where we replaced "rotation", "factor", "left" and "right" with "rot", "f", "l" and "r" for readability.[4]

Augmentation Name	Augmentations used	Highest Accuracy in Validation
Augment	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-color(probability = 1, min-f = 0.5, max-f = 0.5)	0.410587
Augment1	p.skew(probability = 0.5) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-color(probability = 1, min-f = 0.5, max-f = 0.5)	0.413448
Augment2	p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-color(probability = 1, min-f = 0.5, max-f = 0.5)	0.430615
Augment-onlyCrop	p.crop-rnd(probability = 1, percentage-area = .99)	0.9099
Augment-onlyCrop2	p.crop-rnd(probability = 1, percentage-area = .9)	0.9013
Augment3	p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1)	0.9099
Augment4	p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1)	0.9056
Augment0	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1)	0.9084
Augment5	p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5)	0.8026
Augment6	p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-brness(probability = 1, min-f = 0.5, max-f = 0.5)	0.7024

Augment7	p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-color(probability = 1, min-f = 0.5, max-f = 0.5)	0.905579
Augment8	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.2, max-f = 0.2)	0.2375
Augment9	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.01, max-f = 0.01)	0.2303
Augment10	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.75, max-f = 0.75)	0.5866
Augment11	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.85, max-f = 0.85)	0.8169
Augment12	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.95, max-f = 0.95)	0.818312
Augment13	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-brness(probability = 1, min-f = 0.95, max-f = 0.95) p.rnd-color(probability = 1, min-f = 0.5, max-f = 0.5)	0.832618
Augment14	p.skew(probability = 0.5) p.rotate(probability=0.7, max-l-rot=10, max-r-rot=10) p.crop-rnd(probability = 1, percentage-area = .9) p.flip-rnd(probability = 1) p.rnd-contrast(probability = 1, min-f = 0.5, max-f = 0.5) p.rnd-color(probability = 1, min-f = 0.5, max-f = 0.5)	0.702432

Selection of Augmentations As it is important to have a high accuracy on both data sets (precibake and fieldtest) we cannot use an augmentation which best epoch did not have an accuracy better than 70 per cent. Therefore, we choose only those augmentations which have an accuracy better than 70 per cent and which increase the variance of the model. This means that we also exclude augmentations which do not change the color, brightness or contrast of a picture.

Hence, we will in the following only use the Augmentations "Augment5", "Augment6", "Augment7", "Augment11", "Augment12", "Augment13" and "Augment14".

Illustration of Augmentations We will now show eight images of every augmentation with which we trained the ResNet later. Here we see that every augmentation looks different as well as each of image within the augmentation has differences to its neighbor. Still, they were all augmented from the same image of the class "Apfelecke". So, we see in each augmentation a variance of different angels, skews and crops of the original picture, depending of the augmentation we look at. There is a difference in the light conditions like contrast, brightness and color between different augmentations as well but not within an augmentation. Different augmentations may therefore not have more variance in light conditions but the changes of light conditions may cause domain Y to be closer to domain X . See 6.2 for ideas to include variance in light conditions as well.



Figure 16: Original image of Apfelecke

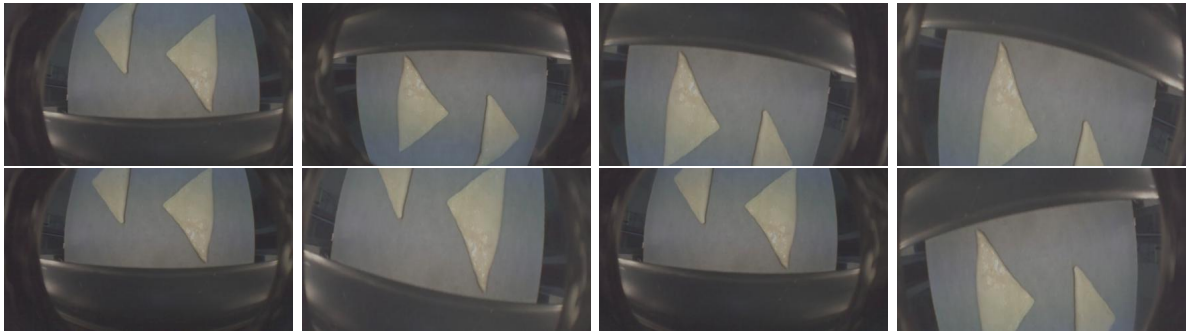


Figure 17: Augmentation5 of an image of Apfelecke

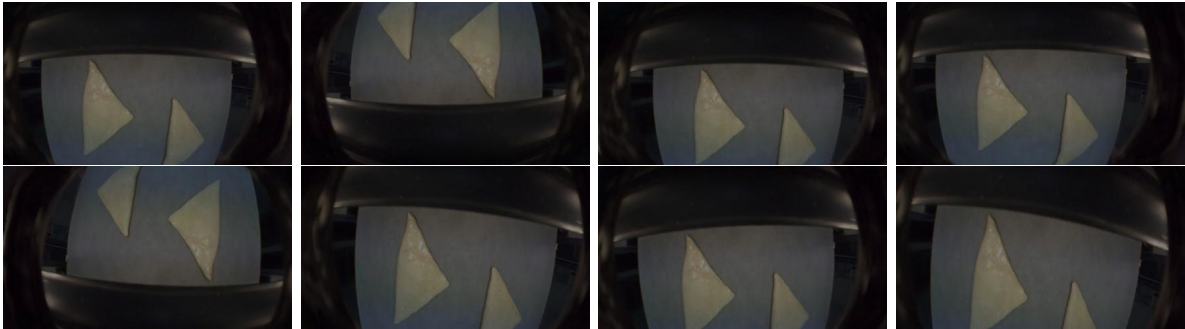


Figure 18: Augmentation6 of an image of Apfelecke

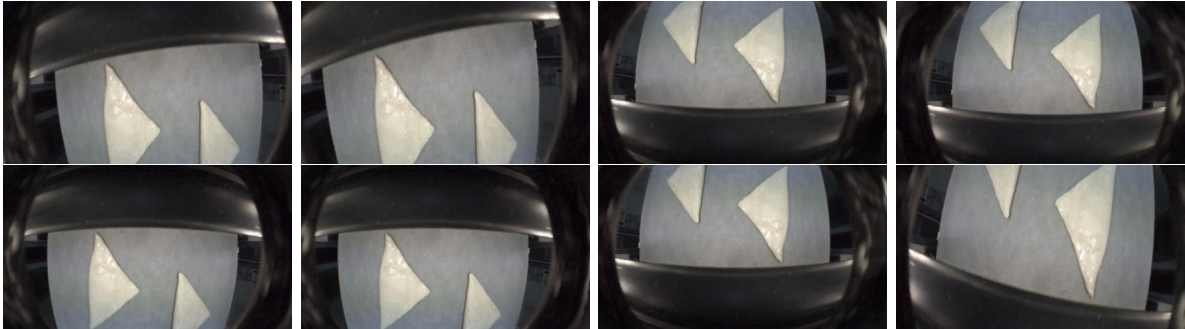


Figure 19: Augmentation7 of an image of Apfelecke



Figure 20: Augmentation11 of an image of Apfelecke

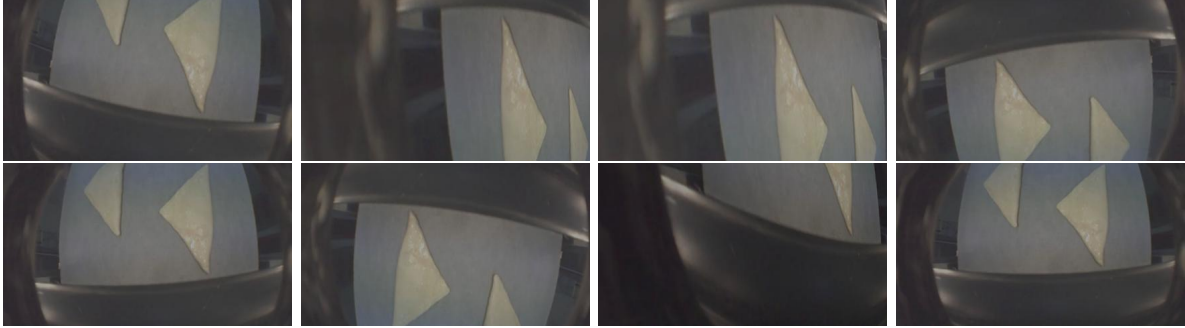


Figure 21: Augmentation12 of an image of Apfelecke

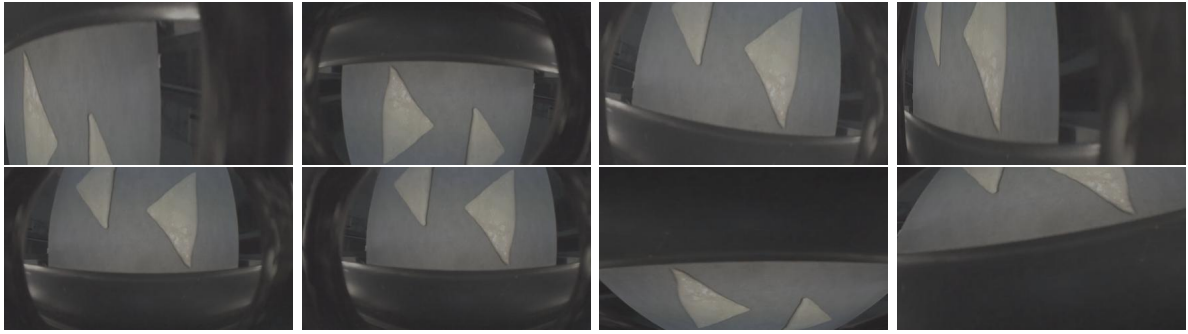


Figure 22: Augmentation13 of an image of Apfelecke



Figure 23: Augmentation14 of an image of Apfelecke

4.2 Choice of Hyperparameters

GcGAN Geometric consistency leaves two hyperparameters to be tuned. Either can the geometric transformation or λ_{geo} be changed. Therefore, we chose the following models to be trained:

Table 2: Hyperparameters GcGAN

Name	Geometric Transformation	λ_{geo}	Details
rot_90_l10	Rotation	10	90 degrees
rot_180_l20	Rotation	20	180 degrees
per_l20	Perspective	20	Points (0,0),(0,28),(28,0),(28,28) to Points (0,0),(0,26),(26,0),(50,28)
rot_90_l20	Rotation	20	90 degrees
rot_180_l10	Rotation	10	180 degrees
per_l10	Perspective	10	Points (0,0),(0,28),(28,0),(28,28) to Points (0,0),(0,23),(23,0),(28,28)




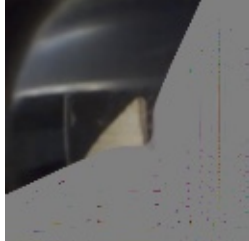

DistanceGAN We have to choose whether we use self or pairwise distance. As mean and variance in both domains are precomputed there is no direct impact of this hyper parameter upon the run time.



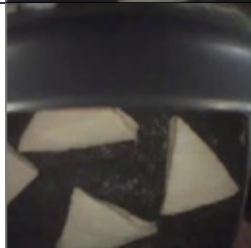
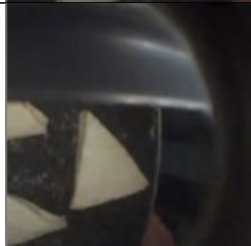
Table 3: Hyperparameters distanceGAN

Name	Details
dis_self	self distance
dis_pair	pairwise distance

Illustration of different Transfer Model Approaches In table 4 we see an image of Apfelecke being transferred to domain Y using different transfer models of GcGAN and distanceGAN. We can see for the model per_l10 that it is important to choose good hyperparameters for the geometric transformation because in this case the geometric transformation uses only have of the images it transforms to.

Table 4: Image from Apfelecke Transferred using different Models

Model Name	Picture	Description
fieldtest		Original image of Apfelecke
dis_self		Image of Apfelecke transferred with distanceGAN
dis_pair		Image of Apfelecke transferred with distanceGAN
per_l10		Image of Apfelecke transferred with GcGAN using a perspective Transformation
per_l20		Image of Apfelecke transferred with GcGAN using a perspective Transformation

rot_180_110		Image of Apfelecke transferred with GcGAN using a rotation
rot_180_120		Image of Apfelecke transferred with GcGAN using a rotation
rot_90_110		Image of Apfelecke transferred with GcGAN using a rotation
rot_90_120		Image of Apfelecke transferred with GcGAN using a rotation

UNIT The UNIT framework produce two generators in each direction of transformation, abbreviate as $x2y$ and $y2x$. We can observe from the life cycles of the networks saved to deduce what the network internally trying to optimise. The most noteworthy difference between training space and testing space is that the training space has cold blue lighting while for the test space, the lighting is warm orange like.

In the first 10000 iterations, the generators trying to add random light blue lighting to mimic the hue of the photos in the original domain. From the 20000 to 50000 iterations, the colour of the generated graph is smooth and do not have abrupt change in lightness. From 50000 iterations onward, the generator also tries to mimic the target distribution of the bread in the pictures by adding or removing bread of the same kind. This behaviour should be avoided as the model should devote to learn only background environmental changes.

Just like most neural networks, the learning rate and the number of epochs greatly influence the performance of the out-coming performance. In the unit models, the parameter logiter controls the frequency that the current generators is being recorded. When the training has just started and the weight has just been initialised, it can be observed that our model is trying to mimic the blue lighting in the training set. Our model shows the typical behaviours of under-performing.

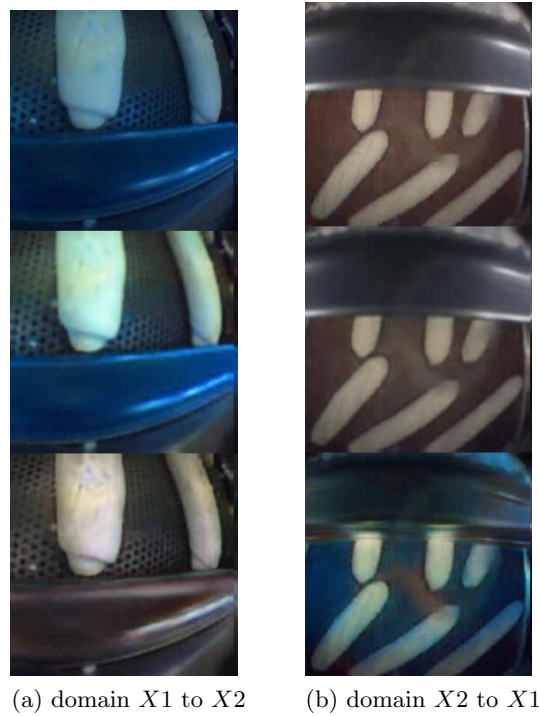


Figure 24: Under-performing Examples of the UNIT model

Over-fitting also occurs in image translation in a different form. A typical observation when over fitting occurs is the network tries to remember not only the environmental parameters, but also tries to manipulate the object distribution. In our case, we want the network to be able to translate the light conditions from domain X to domain Y , but the network should leave the object position intact since this feature is irrelevant in performing object classifications.



Figure 25: Over-fitting Examples of the UNIT model

Below is the table for the hyper-parameters used in the UNIT model:

Table 5: Hyper-parameters UNIT model

Name - Iteration Numbers	step	Comments
iter_10000	10000	under-fitting
iter_20000	10000	-
iter_30000	10000	-
iter_40000	1000	-

Discriminator for Classification We have to choose the weight factors of all individual weight terms. We stick to the weight factors outlined in the papers who introduce the respective methods. For the generator we choose the standard cycleGAN setting:

Table 6: Hyperparameters Generator

Name	value	Comments
loss_identity_A	5	identity loss domain X
loss_identity_B	5	identity loss domain Y
loss_GAN_A2B	1	GAN loss domain X to Y
loss_GAN_B2A	1	GAN loss domain Y to X
loss_cycle_ABA	10	cycle loss domain X to X
loss_cycle_BAB	10	cycle loss domain Y to Y

and for the discriminator we add the loss term and optionally a VAT loss term 7.

Table 7: Hyperparameters Generator

Name	value	Comments
loss_GAN_A2B	1	GAN loss domain X to Y
loss_class	1	cross entropy loss of kind of pastry
loss_class_vat	1	LDS with respect to kind of pastry

Virtual Adversarial Training Here we can choose which size of distortion ϵ and which number of power iterations to compute the LDS we want to perform. We decide to use the default values $\epsilon = 2.5$, $x_i = 1e^{-6}$ and $num_iters = 1$, where ϵ is the magnitude of maximal acceptable noise, x_i the start vector of the power iteration and num_iters the number of iterations in the power iteration. Furthermore opted to not to impose the adversarial training in each iteration but rather every n -th iteration, which reduces the required training time significantly. We have chosen $n = 5$.

5 Results

In the following section we firstly present our results. We used the previously described models to map the data from the fieldtest domain to the precibake domain. Subsequently we use the pretrained ResNet and report the classification accuracy. A natural benchmark is the classification accuracy of the ResNet on the unlabeled fieldtest data without any augmentation that is drawn in 26 as horizontal line.

Surprisingly, ResNet does not have a higher accuracy on the classification on of any transfer of the fieldtest images. Thus, we used augmentation. Here, we see different results regarding different augmentations. When using Augment6 and Augment7, the UNIT model transfers (model 10 - 13) seem to perform much better than the model transfers, which did not use a latent space. For Augment5, Augment11, Augment12, Augment13 and Augment14 the models, which did not include a latent space seem to perform better (model 1-9). This may be caused by a change in the contrast of the images (which is done in Augment5, Augment11, Augment12, Augment13 and Augment14). The difference of the performances depends again on the augmentation and model we look at. In Augment5, Augment7, Augment12 and Augment14 ResNet performs better on the transferred images of the best performing transfer models than it does without using an augmentation on the not transferred images, e.g. the fieldtest domain. Also, in Augment5, Augment12 and Augment14, ResNet performs better on the fieldtest domain without using a transfer model. Non negligible is nonetheless, that except Augment7, ResNet performs worse on the precibake domain, on which it is trained (or on its augmentations), than it does when we do not consider an augmentation.

In figure 27, we only tested the performance of ResNet with one additional augmentation, e.g. Augmentation5. But this time we used a different loss function for our model, e.g. the VAT loss function. (see 3.4). In the figure we can see that UNIT does not work properly under this classification model with and without an augmentation. Also, the threshold is lower as the model performed worse on the fieldtest domain without using an augmentation, but it gets better results in when it uses Augment5. This is the highest classification accuracy of any model on the fieldtest domain. Nonetheless, no transfer model could improve the performance of the ResNet model.

6 Conclusion

Domain mapping Our Semi-supervised Learning approach yielded to better results on the fieldtest domain. This was achieved by different transfer models and augmentations. Whereas some augmentations may cause the classification network to perform worse on the domain on which it is trained, it may lead to better results combined with the transfer model. Nonetheless, we still need to tune the hyperparameters of the transfer models as the accuracy only got slightly better or even worse depending on the augmentation. Using VAT gave us a higher improvement when we trained the ResNet model with augmentation Augment5, but no transfer model could improve it anymore.

We investigated several circular domain mapping methods, e.g., cycleGAN, GcGAN and distanceGAN. We transferred unlabeled pictures from the fieldtest domain and classified them subsequently with the ResNet trained from the precibake domain. Comparing the classification accuracy of the ResNet on the translated images we find that circular constraints yield an improvement in the classification accuracy compared to the plain ResNet for several augmentations. We observed no significant difference from using self- or pairwise constraint distanceGAN. The only difference as a differently long precomputation time. Thus, we can conclude that both are a legitimate ways to enforce a circular constraint, however without specific advantages. Also the different hyperparameter settings of GcGAN lead to comparable results. It can as such be concluded that introducing a circular constraint and as such transferring the pictures improves training accuracy, however the hyperparameters of the chosen GAN have only a minor overall impact.

UNIT model is also a different form of the GANs with more designed structure. Over-fitting is more noticeable in this model, when the iteration does not terminate after the network learn how to transfer the environment variables. Theoretically, such a behaviour is expected as environment variables are first to learn due to the huge impact they have on our data. We should prevent the networks from alternating the locations of objects, by choosing a good iteration number from experiments. Aside from iteration number, the step size is also another number we should attend to. Currently, the step size does not change the behaviour of our image translation network. When choosing a small step size, we can observe what the network tries to do in each step, and is desired when we intend to debug the whole process.

We found that the chosen augmentation has a huge influence upon the observable classification accuracy. Here, we saw that a change in colour, brightness or contrast influences the accuracy a lot.

Using the Discriminator for classification We found that our simple summation of constraints did not result in a desirable performance. This underlines the fact that GANs are rather hard to train, thus the composition of several losses might need further hyperparameter tuning.

6.1 What has been learned?

A first immediate observation after the start of the project was the urgent need to use GPU's. However even using the Precibake GPU's we encountered very long training times for the mapping between domains. Thus it was desirable to pay attention to every detail increasing the experimental speed. A major speed up was the decision to use UNET as generator in the cycleGAN settings. As this network increases the size of the resulting figure this yields particularly at the beginning of training shorter episodes. We further encountered the need to store pre-computed results, for example the domain mean and variance for the distanceGAN. Another method to speedup training was to compute the adversarial loss only every 5-th epoch. One lesson we experienced are that GANs are hard to train. This is particularly true if we combine different loss terms from different models where the weight terms of each individual term might no suitable combination. We also found that the quality of labeled data is extremely important.

6.2 Future Work

There is still a lot of tuning of the hyperparameters necessary for our transfer models to perform better. This can be done especially in the case of GcGANs as the performance of this model transfer may depend on the geometric transformation which is chosen. If we are able to have a generator which is invariant under a specific geometric transformation, which may be similar to the mapping from domain X to domain Y this would yield to much better results in the accuracy.

As the results depend a lot on the augmentation, we should also focus on finding a better augmentation, which gives the data the right amount of variance. Therefore, we should expand the variance of color, brightness and contrast.

As the best test result was achieved using VAT, we should also work more with it and try different augmentations while using VAT as well as different transfer models to determine the VAT loss.

Nonetheless, we can add more data to train a better classifier and to train a better transfer model, which does not yet improve the accuracy without using augmentation. So if we would add more data with more variance, we may not need to augment the data. Then, we might add more product classes to the data and see if the model still performs well. After adding all needed product classes this model could be used without the need of labeled data. This would result in saving a lot of time. Thus, the customer will get and may be able to use its products earlier.

A further direction of research could be to tune the weight factors of the loss terms in the discriminator for classification setting. In our work we always used the picture space for classification, either with the ResNet or the discriminator in the cycleGAN setting. However, the UNIT models has the shared space assumption between both domains, which applies directly to our problem. Thus a future direction of work is to perform classification in the latent space.

So far, we were given only two domains, whereas PreciBake has many different ovens in many different locations. Thus an expansion of our approach to a multi domain adaptation problem, particularly the issue of knowledge transfer between different field test domains is desirable.

References

- [1] Precibake homepage. <http://www.precibake.com/>. Accessed: 2019-02-09.
- [2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.
- [3] Sagie Benaim and Lior Wolf. One-sided unsupervised domain mapping, 2017.
- [4] Marcus D Bloice, Christof Stocker, and Andreas Holzinger. Augmentor: an image augmentation library for machine learning. *arXiv preprint arXiv:1708.04680*, 2017.
- [5] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, Kun Zhang, and Dacheng Tao. Geometry-consistent adversarial networks for one-sided unsupervised domain mapping. *arXiv preprint arXiv:1809.05852*, 2018.
- [6] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [7] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *corr* (2015).
- [8] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *CoRR*, abs/1412.6572, 2014.
- [9] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017.
- [10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.
- [11] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.
- [12] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks, 2017.
- [13] Takeru Miyato, Shin-ichi Maeda, Shin Ishii, and Masanori Koyama. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2018.
- [14] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, Oct 2010.
- [15] Anthony L Peressini. *Mathematics for high school teachers: An advanced perspective*. Prentice Hall, 2003.
- [16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

- [17] Aitor Ruano. Pytorch-cyclegan. <https://github.com/aitorzip/PyTorch-CycleGAN>, 2017.
- [18] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.
- [19] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.
- [20] Gerard Venema. *The foundations of geometry*. Pearson Prentice Hall, 2006.
- [21] Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.
- [22] Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect, 2018.
- [23] Leland Wilkinson. *The grammar of graphics*. Springer Science & Business Media, 2006.
- [24] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

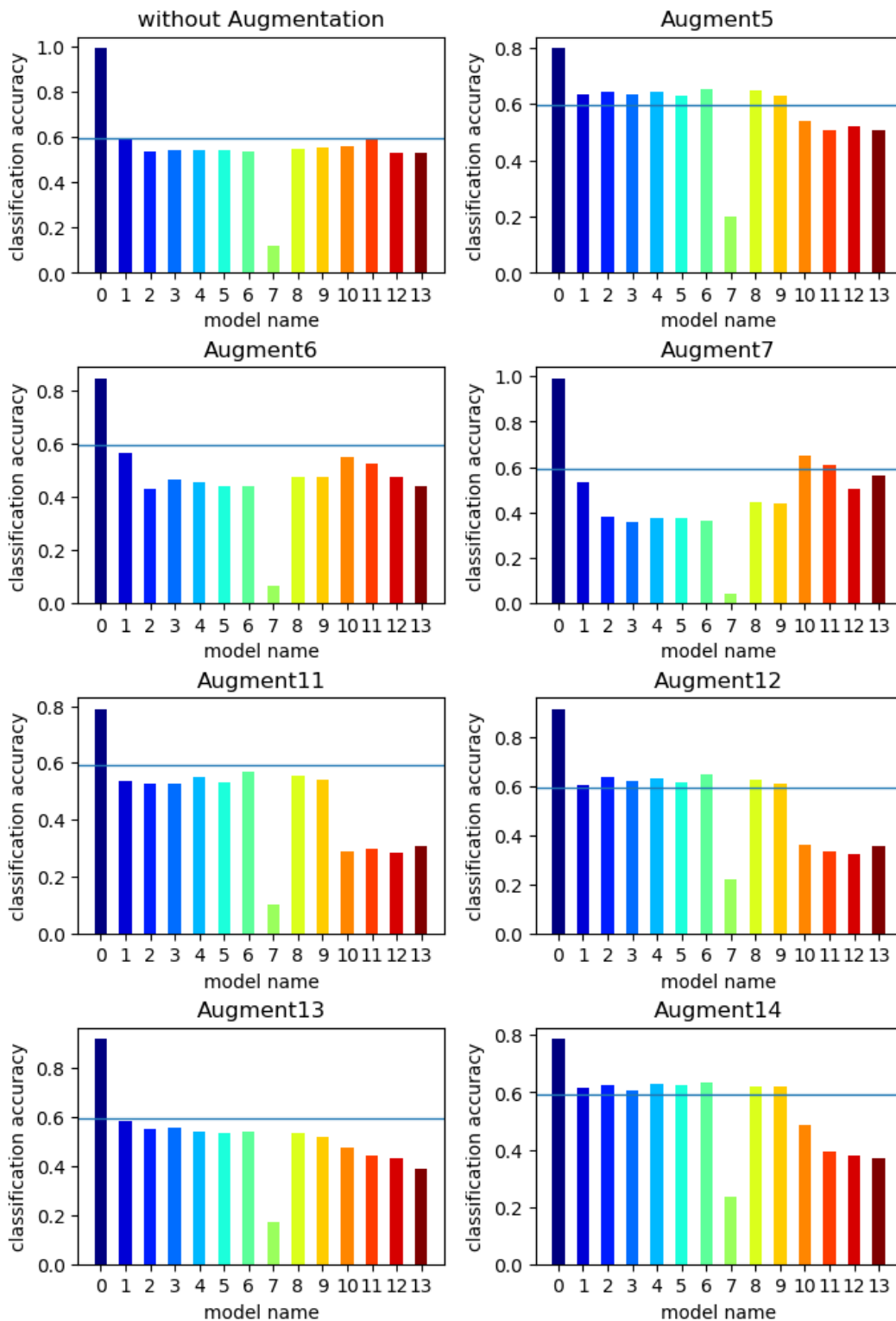


Figure 26: Classification accuracy of the, with different augmentations, pretrained ResNet for different mapping models between fieldtest and precibake domain.

0 - domain Y , 1 - domain X , 2 - rot_90_l10, 3 - rot_180_l20, 4 - per_l20, 5 - rot_90_l20, 6 - rot_180_l10, 7 - per_l10, 8 - dis_self, 9 - dis_pair, 10 - iter_10000, 11 - iter_20000, 12 - iter_30000, 13 - iter_40000

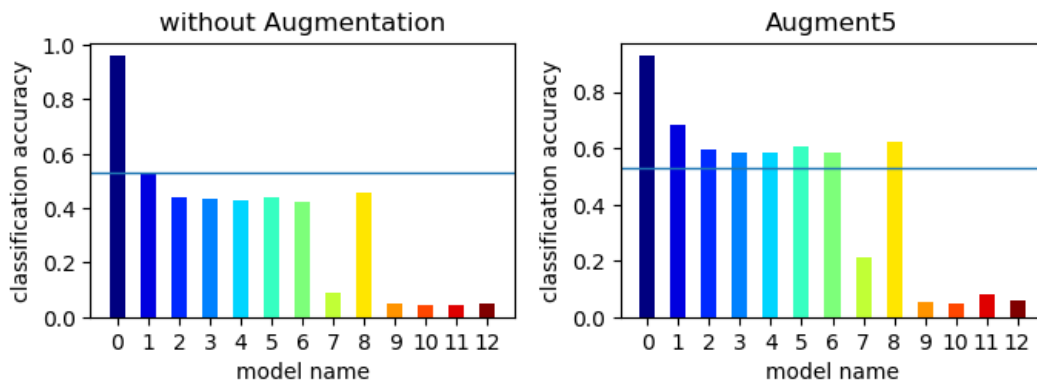


Figure 27: Classification accuracy of the, with different augmentations, pre-trained ResNet for different mapping models between fieldtest and precibake domain using VAT.
 0 - domain Y , 1 - domain X , 2 - rot_90_l10, 3 - rot_180_l20, 4 - per_l20, 5 - rot_90_l20, 6 - rot_180_l10, 7 - per_l10, 8 - dis_self, 9 - dis_pair, 10 - iter_10000, 11 - iter_20000, 12 - iter_30000, 13 - iter_40000