Conditional Generative Adversarial Network for image translation

Hands-on Deep Learning for Computer Vision and Biomedicine

Supervised by Qadeer Khan & Patrick Wenzel

Report by Ahsan Lodhi & Yang An

Chair of Computer Vision & Artificial Intelligence - Prof. Dr. Daniel Cremers

1 Introduction

Autonomous Driving (AD) is one of the first major products showcasing a big impact of artificial intelligence (AI) in our daily life. Safety is crucial for a positive public opinion and the adaption of this and other AI products in society. Autonomous cars must work under all driving environments suitable for the given autonomous level to maintain the safety of passengers as well as pedestrians and other traffic participants.

In this work we address the perception of autonomous cars and make them more robust under different driving environments. For that matter, we train a Generative Adversarial Networks (GAN) to translate an input image into a) another weather domain (e.g. translate sunny -> rainy) b) another camera perspective. Extending GAN to a conditional GAN (cGAN) makes it possible to control the output to a specific weather domain/camera perspective.

2 Conditional Weather Translation

Autonomous cars need to perceive and interpret its environment in order to take the right actions. Real environments are dynamic and changing weather and daylight conditions directly influences the appearance of the whole scene.

In this section, we present our work to translate the weather and daytime conditions of an input image. This can be used for data augmentation and train more robust models for other computer vision tasks. In another way, the network can be used directly and extended for other computer vision tasks.

The goal is to have one network to translate images. The emphasis is on one. First, it is unfeasible to train a network for each pair of translation, as the number would grow quadratically with the number of domains. Similar to multi-task, the network can use the whole dataset to improve the realism of the images, instead of only using a subset of one translation, e.g. $X \rightarrow Y$. Speed and memory are another reason to only use just one model, which is especially a priority in real-time applications on weak hardware such as autonomous driving. To control the output, the user has to give the network an instruction to which domain the input should be translated to. To accomplish this, the network must be conditioned, leveraging a normal GAN into a conditional GAN.

In this report we will use the uppercase letters X, Y to indicate domains. The two lowercase letters are images from the specific domain $x \in X, y \in Y$. Note that other letters can indicate different things which are specified later.

2.1 Method

2.1.1 CycleGAN and StarGAN

Zhu et. al introduced CycleGAN (Zhu et. al, 2016), a universal method to do image-to-image translation. To accomplish this, the network must learn the mapping between the input image x and the desired output image y. The translated image should keep all the content of the input and only change the style to match domain Y. The easiest way would be, to use paired images and train the network with them. Images are paired if the ground-truth image from domain Y is available for an input image from X. An example would be a photo of the Eiffel Tower, taken at the exact same spot during summer (domain X) and winter (domain Y). Unfortunately, paired images are very hard to acquire in real world for many translations. Often it is even hard to define, what the ground-truth image should look like, especially in artistic domains.

CycleGAN tackles this problem and works with unpaired data. Staying with the summer to winter translation example, an unpaired dataset would be arbitrary images of summer in one folder and winter in another folder, instead of photos from the exact same scene. Using a Vanilla GAN with this data, the problem is under-constrained. The discriminator only tells the generator, if it's real or fake. Any convincing fake winter image would be classified as winter. So the generator could use this as a loophole and just generate winter images without keeping the content of the input. The most extreme case would be, the generator just produces the exact-same successful winter image over and over, no matter what the input image is, also called mode collapse. To produce meaningful translations, CycleGAN uses a $Cycle\ Consistency\ Loss$ to constraint the possible output image and keep the content of the input. First a GAN G translates the input x into a fake y' and then another GAN F translates y' back into domain X, resulting in a fake image x'. $Cycle\ Consistency\ Loss$ penalizes differences between x and x' using L1-Loss, so implicitly GAN G will try to produce a fake y' that keeps as much content as possible for easier backtranslation to $x' \approx x$. So, translating our summer image into a winter and back to summer, we should ideally arrive at the same image as the input.

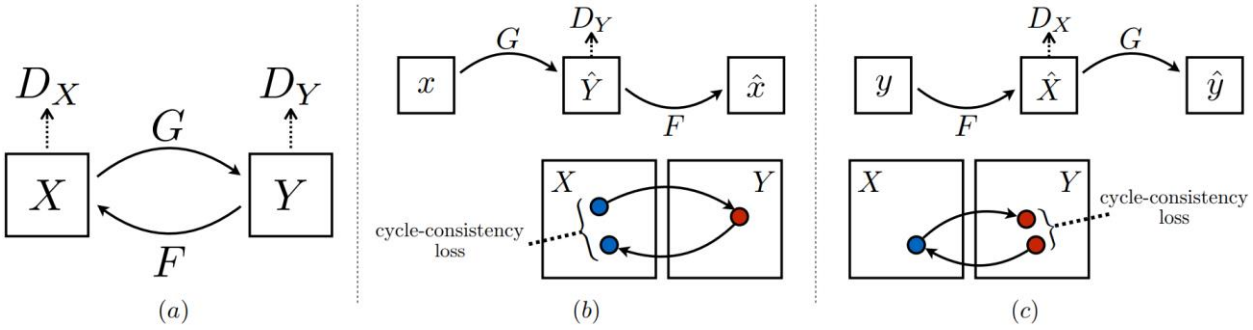


Figure 1: Conceptual Architecture of CycleGAN. (Zhu et. al, 2016)

With many domains, every mapping from one domain to another domain needs to be done with another network. Using more than two domains, the combinations will grow quickly to unfeasible magnitudes, e.g. with four domains, we would already need to train 12 models. In general, if n domains

are translated, $n(n-1)$ networks need to be trained. As stated before, our goal is to use only one conditional model to translate an input image into different domains.

Our idea to implement the conditioning is, to give our desired output domain information in form of extra channels concatenated to the RGB-channels as input (see Figure 2). In the binary case, one extra channel would be enough. Filling the channel with 0s would indicate to translate the input into domain X and filling with 1s a desired translation to domain Y. With more domains, one-hot-encoding can be used, with every element of the i-th channel corresponding to the i-th domain being 1s and the others 0s.

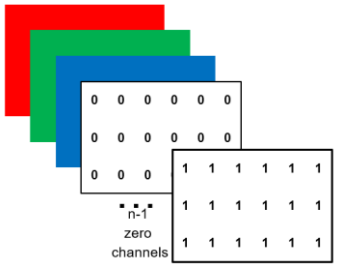


Figure 2: Concatinating a one-hot-encoded conditional channel to the RGB-image.

Besides changing the input, the discriminator also needs to be extended, to give the generator a corresponding supervision signal. The naïve approach would be to build a discriminator for every domain. Even though the discriminator is only needed during training and in application deployment wouldn't increase computations, it is against our philosophy of one model. So, our idea is to condition the discriminator in the same way as the generator, then the discriminator should tell if the image is a real or fake image of the given domain.

We implemented parts of our proposed architecture, before we came across StarGAN (Choi et. al, 2018). It uses the same technique of adding conditional channels. There are two major difference to our proposed architecture. First, StarGAN doesn't condition the discriminator. Rather, the discriminator outputs another supervision signal for the generator besides the real/fake loss D_{src}, called the classification loss D_{cls}. The real/fake loss ensures the quality of the generated images to be looking real. The classification loss makes sure, the image from the generator really belongs to the conditioned domain. Additionally, it uses Wasserstein GAN objective to replace binary-cross-entropy for real/fake determination to stabilize training and creating higher quality images.

As a summary, StarGAN uses three loss functions, the real/fake loss $\mathcal{L}_{adv}$, classification loss $\mathcal{L}_{cls}$ and still the cycle consistency loss to preserve content $\mathcal{L}_{cyc}$.

We adapted the network architecture of the StarGAN paper, which consists of a generator with two convolutional layers of stride two for downsampling, then six residual blocks and afterwards two transposed convolutional layers with stride 2 for upsampling. The discriminator is a PatchGAN, which only uses patches (i.e. cropped out parts from the whole image) to increase speed and improve the texture. The details of the network can be found in the original StarGAN paper.

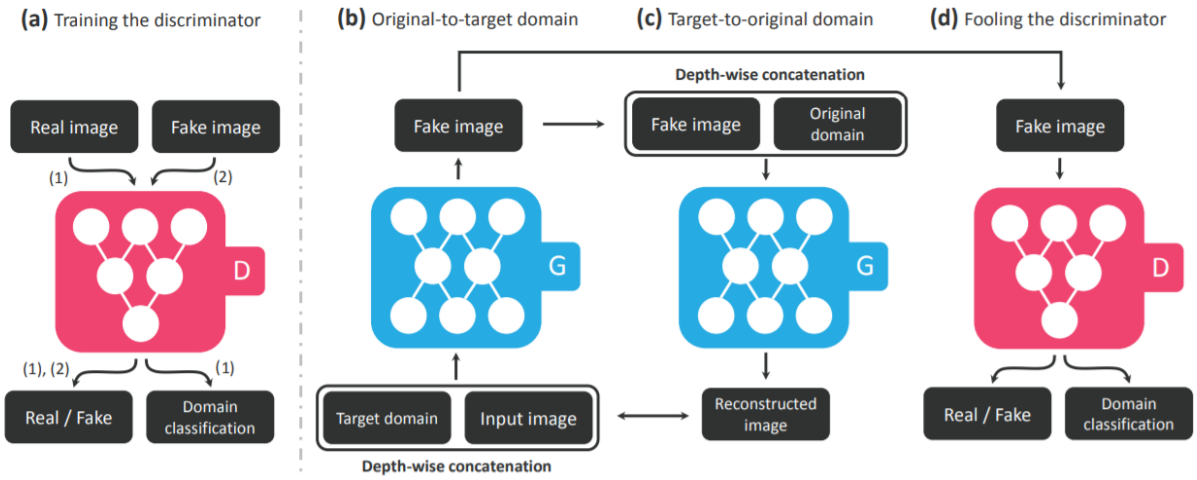


Figure 3: StarGAN Architecture, the main extensions to CycleGAN are the conditioning of the input and the domain classification of the discriminator. (Choi et. al, 2018)

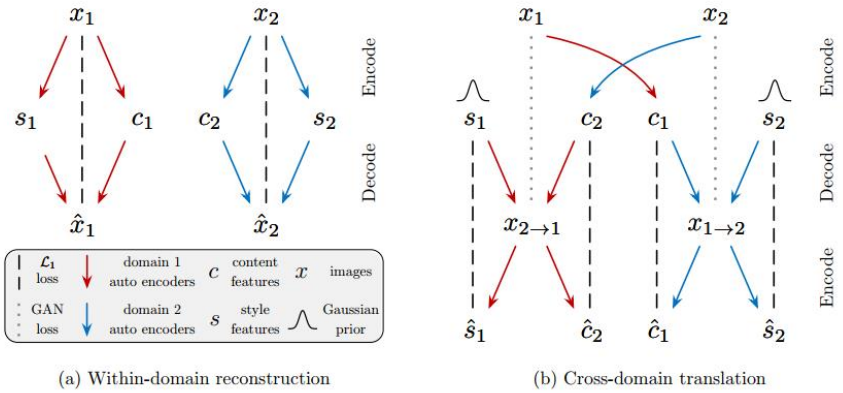
2.1.2 MUNIT

Nvidia introduced another approach for image translation Multimodal Unsupervised Image-to-Image Translation (MUNIT) (Huang et. al, 2018). Instead of using a cycle consistency loss to preserve the content of the input image, it decomposes the image into a style vector and content matrix. The style vector is specific for every image and similar within a domain. The content matrix contains the structures and objects shared throughout all domains. Using the content matrix of image x and the style vector of image y we can translate images from domain X into domain Y.

The two main differences to CycleGAN / StarGAN is its multimodal assumption and unsupervised setting. StarGAN is unimodal, which means each input image and conditioning will result into one deterministic output image. In the real world, within a domain, there are different possibilities to translate the input image. As an example, for the task of translating dogs into cats, there are multiple output possibilities as there are different breeds of cats. With MUNIT, we can use the style code of a certain breed to control our translation finer.

Thanks to the unsupervised nature of MUNIT, it has less requirements for the dataset compared to CycleGAN / StarGAN. Beyond unpaired images, MUNIT don't need any labels at all, as the decomposed style code acts as target conditioning.

The architecture consists of an encoder and decoder to form an autoencoder. For each domain a new model is trained. As an autoencoder, the model is trained to learn a mapping of the input to the same input again. This is enforced by minimizing L1 loss between input and output image. Furthermore, to ensure that the content matrix contains all the shared elements and structures between the domains, the content matrix of x and a prior style Gaussian style vector are feed into the decoder and encoder and the L1 loss of original and output content matrix/style vector is calculated. Note, that the authors use $x_1 and\ x_2$ instead of $x\ and\ y$.



(a) Within-domain reconstruction (b) Cross-domain translation

Figure 4: Architecture of MUNIT. (Huang et. al 2018)

To use MUNIT for our purpose, we only want one model again. We can treat all different weather settings as one domain. To control our output, we need to provide the model an input image x and a guide image y from our desired weather setting. From the input image we extract the content, while from the guide image the style vector. Feeding both into the decoder, we will get the translated image.

2.2 Experiments

2.2.1 Dataset

For training and evaluation of weather translation, we relied on two different datasets. To build a working prototype, we used simulated data from CARLA. 10 different weather conditions are used, differing in color, dry/rain/wet and other details. The dataset consists of 5000 images from each domain, resulting in 50,000 images in total. They have a size of 128x128. The images are also unpaired to ensure better future usability. The images are continuously taken while the virtual car drives around. Therefore, the images are sequential.

Besides the simulated dataset, we used the Oxford Robotcar Dataset with real images. The researchers drove around the same route and captured images of the same scene under different weather and lighting conditions. We've picked four different weather/lighting conditions: Sunny, night, rainy, snowy. Our subset consists of 1000 images from each category. As the data is made available only in raw format, a script to convert them into jpg-format has been written. Also here, the images are sequential.

2.2.2 Training

We've implemented the models using the codes provided by the respective authors and modified it to our needs. For StarGAN we trained it for 40,000 iterations on the simulated dataset with batch size 64. The Oxford Robotcar Dataset is more complex, therefore we trained it longer for 200,000 iterations with batch size 64. The other hyperparameters follows the default settings.

MUNIT was trained for 120,000 epochs with a learning rate of 0.00001. Again here, we followed the default hyperparameter settings.

2.3 Results

MUNIT didn't worked as well for weather translation. As seen in Figure 5, the translated image only worked for the green target domains (w3 and w4). The other translations didn't seem convincing. We believe that the style code conditioning is under-constraining. The style code of two different weather domains are too similar and results in an overlapping translation. Therefore, we depicted MUNIT and only continued with StarGAN for real images.

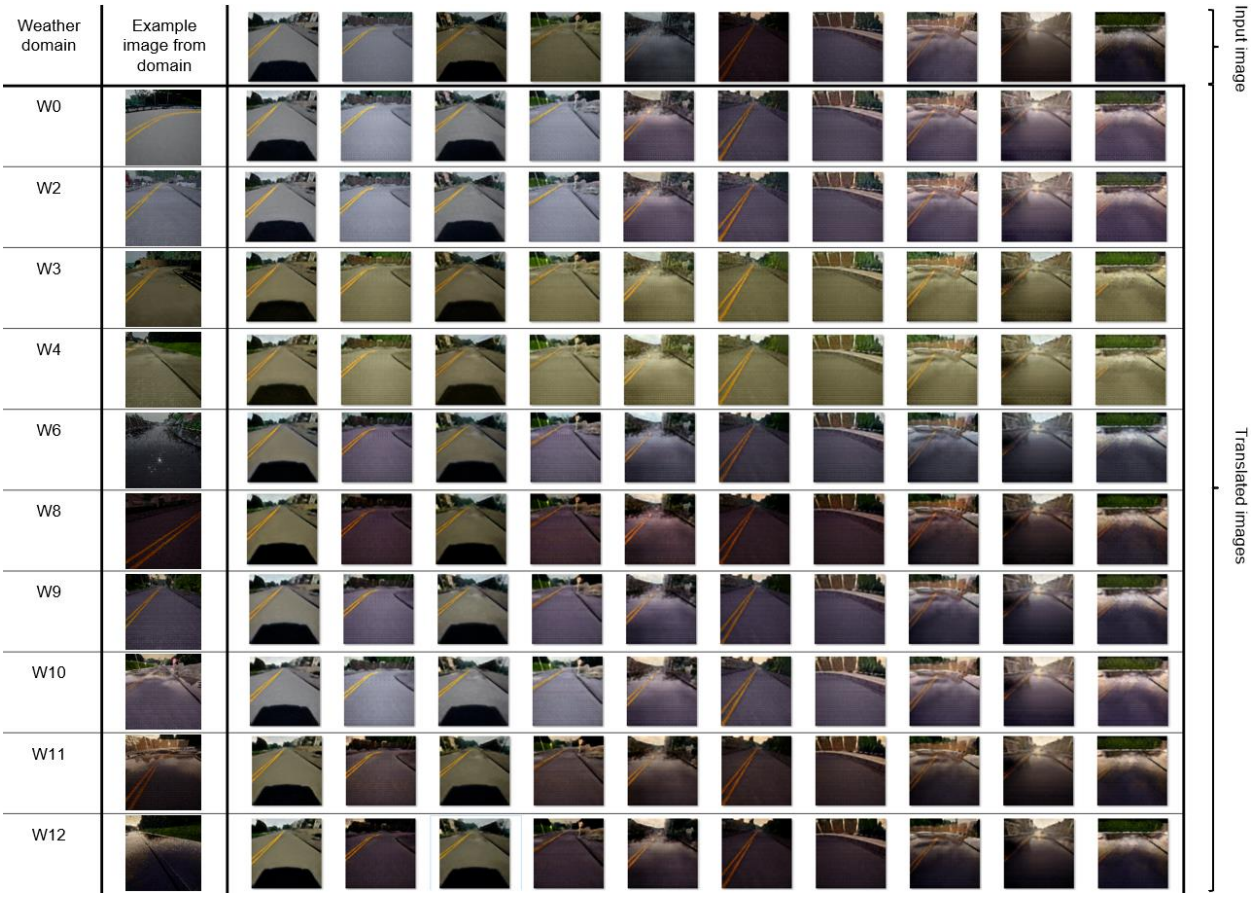


Figure 5: The results of MUNIT. The images in the first row are the input images, one from each domain. The example images in the second column provide the target style. The other images are the translated images, with the column indicating the input image and the row the target domain.

StarGAN works a lot better to translate the images into another weather domain. The first results on simulated data can be seen in Figure 6 and Figure 7.

Figure 6: Results from StarGAN on simulated data.

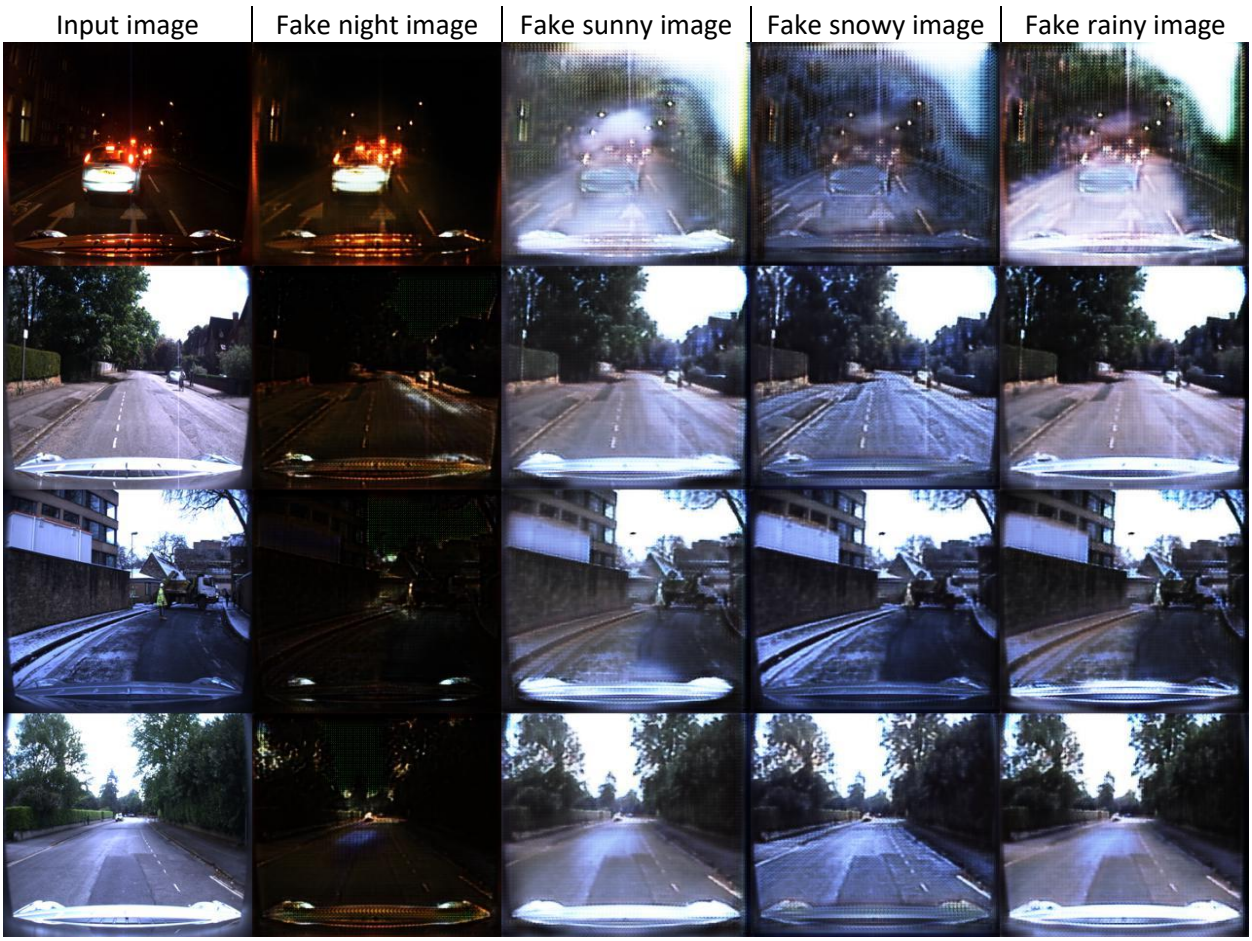


Figure 7: Results from StarGAN on the Oxford Robotcar Dataset.

3 Camera Shifting

After image translation with different weather settings worked well with StarGAN, our task changed to shifting the camera perspective. Imagine cameras mounted along the lateral axis of a car. Given an image x from camera X, how would the image look like when camera Y (mounted 1m to the right) would have taken it? This could be useful as redundancy or for sensor fusion, when one camera lens is dirty or the car in the front blocks the view of one camera. The task is related to weather translation. While weather translation has fixed domain labels (multi-class problem), camera shifting uses continuous variables as target label. We will give the model an arbitrary image from any camera and a value to shift the camera along the lateral axis, no matter where original camera was. The goal is to let the network learn the concept of shifting the camera by a certain amount, even though it didn't learn those values explicitly during training.

3.1 Method

We continue to use StarGAN for perspective shifting and change the architecture from multi-class to continuous variable. Instead of using one-hot-encoded extra channels for conditioning, only a single channel is added to the RGB-image. This conditional channel contains the value to shift the camera position virtually along the lateral axis. For the discriminator each camera is still treated as a class, so it is kept the same. In addition, we exploit advantages of our paired/aligned dataset. In an aligned dataset, the desired ground-truth output image is already given, so we know exactly how the network should translate the image. To keep the content of the scene and only translate the style, we rely on L1-Loss instead of cycle consistency loss, which should constraint the output better resulting in better quality.

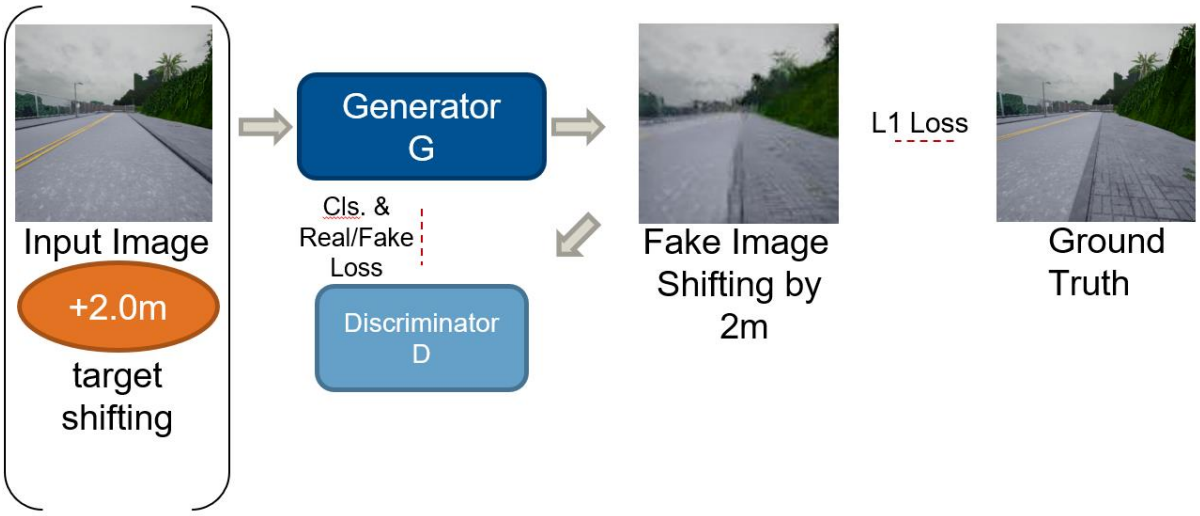


Figure 8: Our proposed architecture for camera shifting. The input is the concatenation of image and target shifting. The fake image is then evaluated by the discriminator for its general realness and camera class. Additionally, the fake image is compared to the ground truth, resulting in a L1 Loss.

3.2 Experiments

3.2.1 Dataset

Again, we are using CARLA simulator to generate the dataset. Our first dataset consists of images taken from six different cameras. When 0m is the middle of the car and positive values indicates a shifting to the right in the direction of driving, the cameras had the following positions: [-1.3m, -0.3m, 0m, 0.5m, 1.0m, 2.0m]. For every camera, 4000 images are taken, resulting in 24,000 images. We've split the dataset into a training dataset consisting of 3600 images of each camera and the rest are used for testing. As the images are sequential and two consecutive images are looking very similar, we've split the data so that training and test dataset are two different sequences.

For the first dataset, the simulated car drove perfectly on the road, always in the middle of the lane. Thus, the images from one camera look very similar from a perspective view, e.g. the middle and pedestrian lane is always in the same position. To ensure, the network learned the concept of shifting, instead of just overfitting and memorizing how images from a certain camera looks like, we generated a second dataset. This time, the car didn't drove perfectly, rather the car drove a serpentine line. The camera position and number of images stayed the same. We used a randomness parameter of 0.1 in CARLA.

3.2.2 Results

We first trained our model for 100,000 iterations on our first dataset. The results can be seen in Figure 9. It works quite well on cameras the model has seen during training.

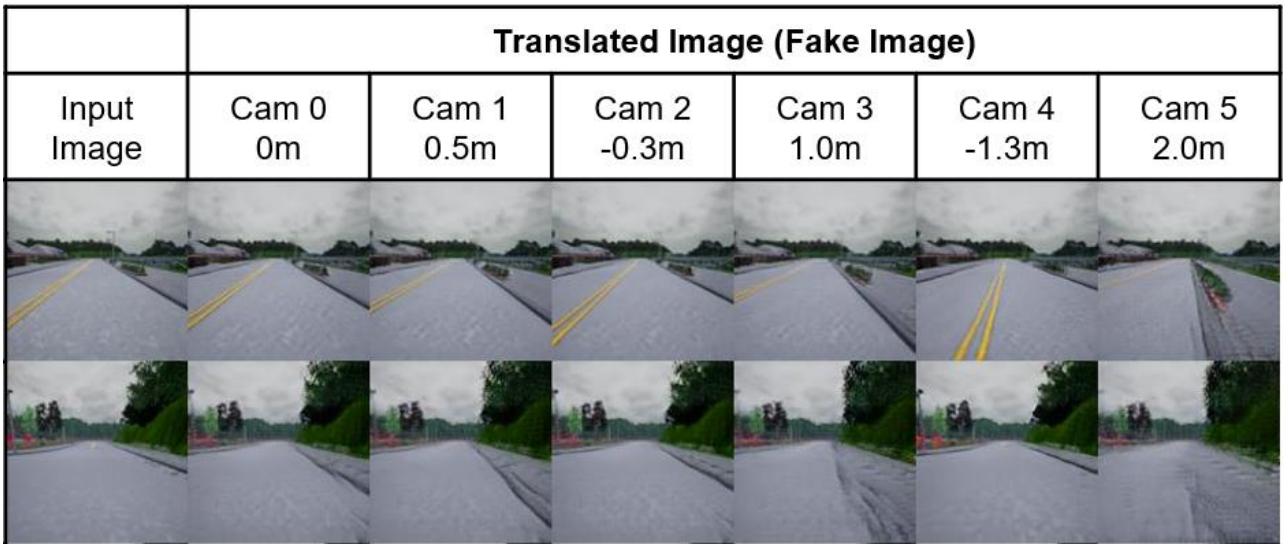


Figure 9: Results from our first dataset.

Afterwards, we tested our model on our second dataset, where the car drives in serpentine lines. At the same time, we tested to shift the image to unseen camera positions during training. This also works well indicating, the model really learned the concept of shifting cameras, instead of only memorizing.

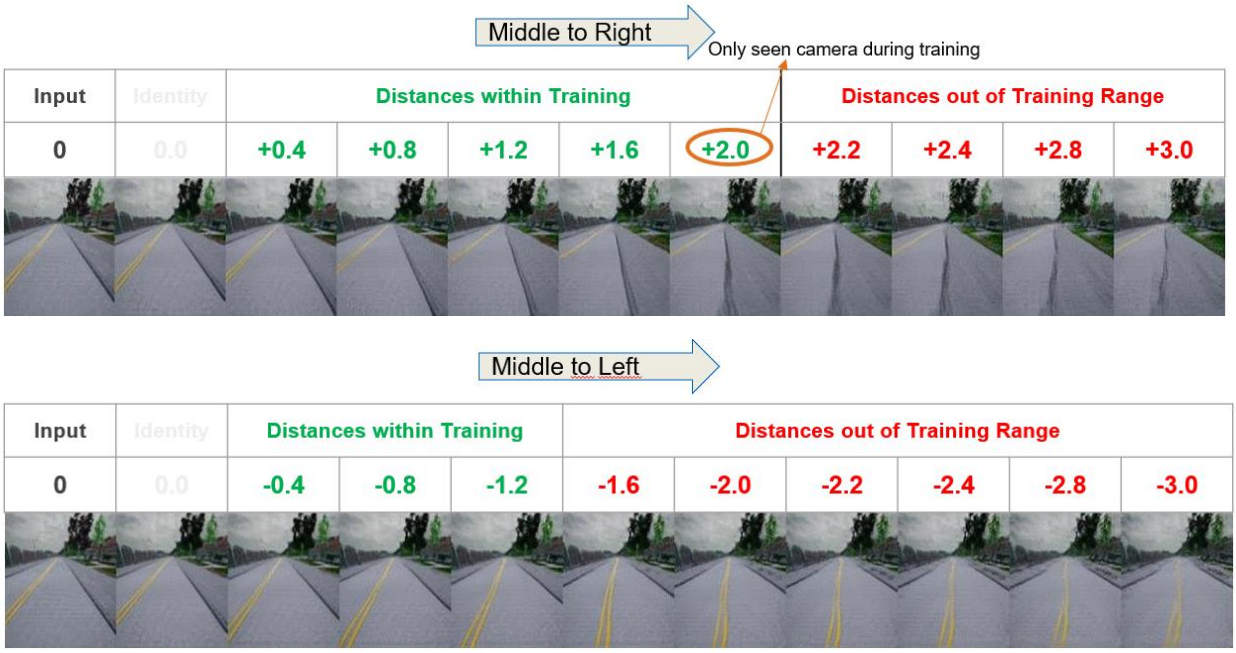


Figure 10: Output of our model for the second dataset, containing images where the car drove in serpentine lines. All values expect for +2.0m are unseen during training. Additionally, many values are completely out of training range. The results indicates the network learned the concept of shifting cameras by a certain value, even for out of range value close to the training interval.

3.2.3 Experiments with different loss functions

To determine the impact of different loss functions to the whole architecture, we used different combinations of loss functions. We always trained the model for 100,000 iterations to ensure comparability.

In the first test, we compare a pure GAN (no L1 loss), with an autoencoder style architecture (only L1 loss). We can see the results in Figure 11 and Figure 12.

A pure GAN without L1 loss works as well as with it. The difference in the results are not possible to see with eyes. This indicates either, that the value of L1 loss is comparably small to the other losses and gets neglected during training. Or there is redundancy between L1 loss and real/fake plus classification loss.

With only L1 loss, the translated images are still looking correct. But the images obviously lack details and are blurrier compared to GAN.

Pure GAN (classification and real/fake loss)

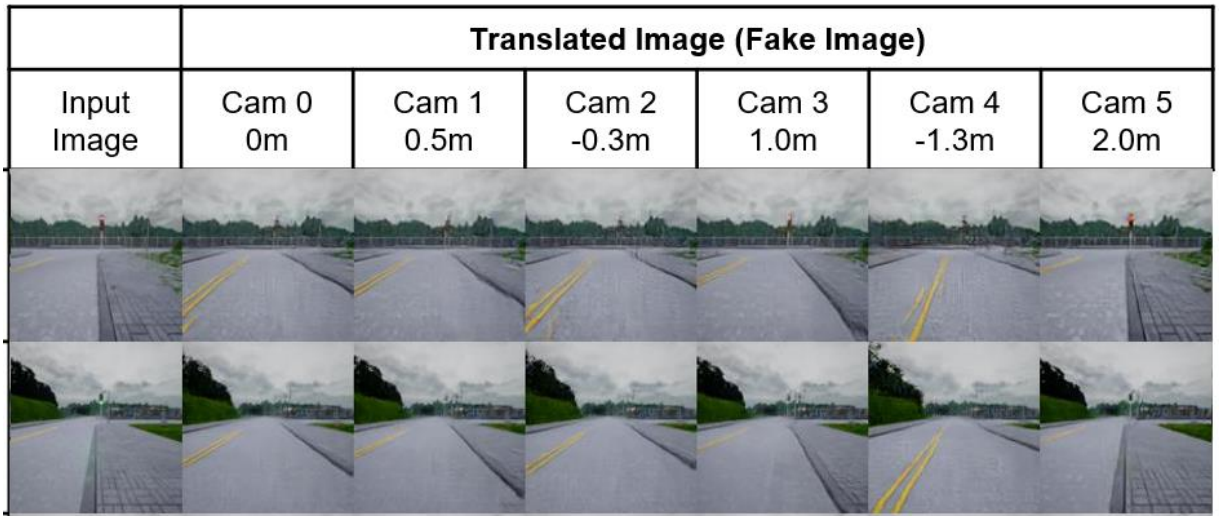


Figure 11: Pure GAN, the results are the same (evaluated with eyes) as with additional L1 loss.

Only L1

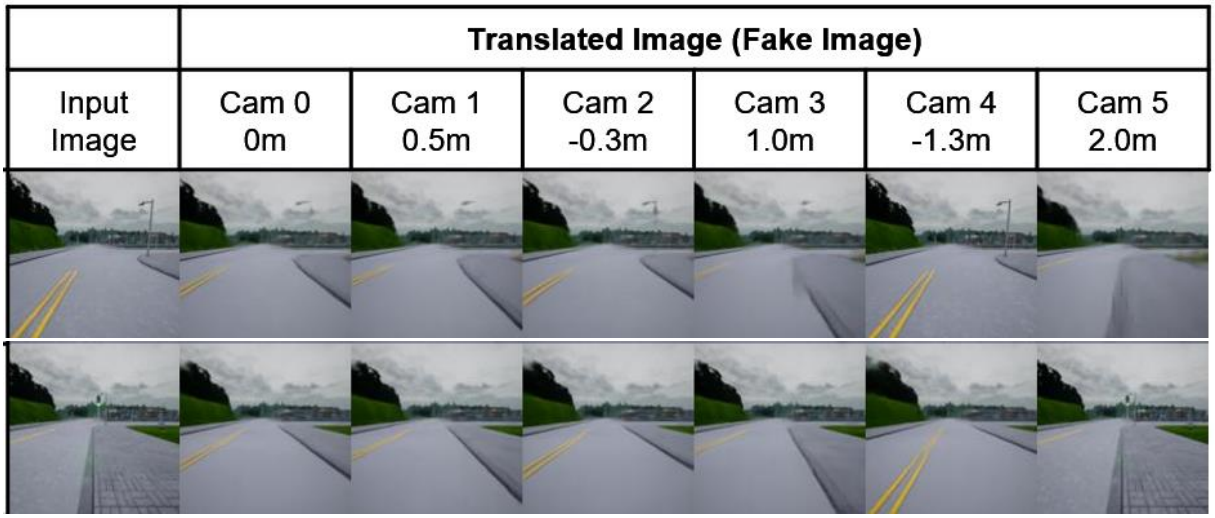


Figure 12: Only L1 Loss, the images can still be shifted correctly, but the images are blurrier than with a GAN.

Using only classification loss or real/fake loss respectively, we get some more obvious results. As seen in Figure 13 only using classification loss, the images look far from realistic, with almost no details. Even though this is obvious, it is interesting to evaluate the discriminator and see with how little information the classification is based on.

With only real/fake loss, the generator found a loophole by keeping the input image as output, as the most extreme form of mode collapse.

Only classification loss

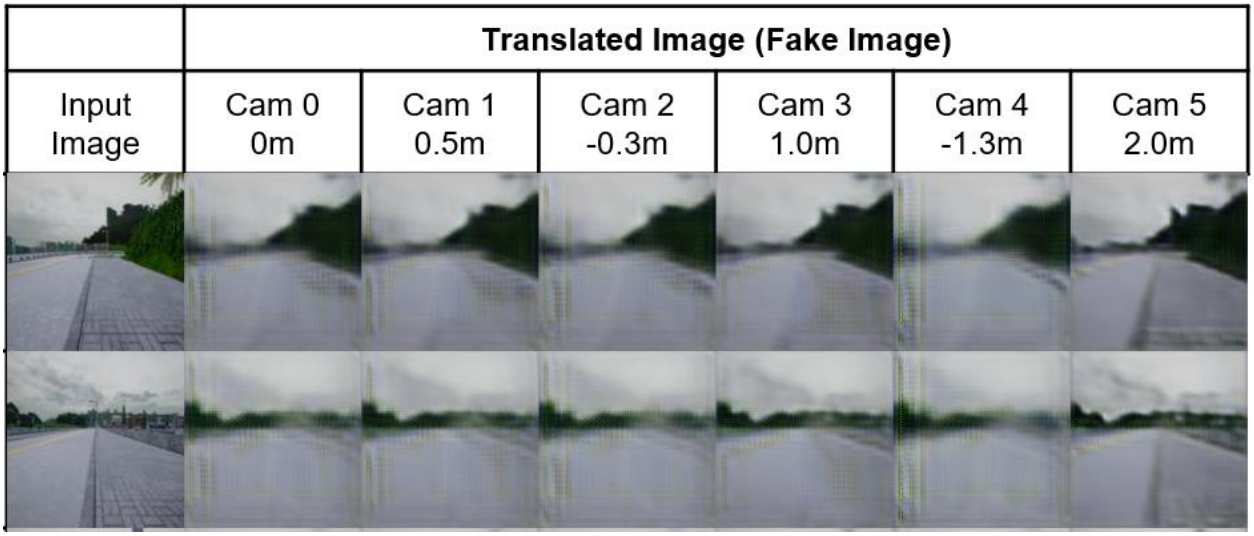


Figure 13: Only classification loss, we can see with how little information the discriminator classifies the images.

Only real/fake loss

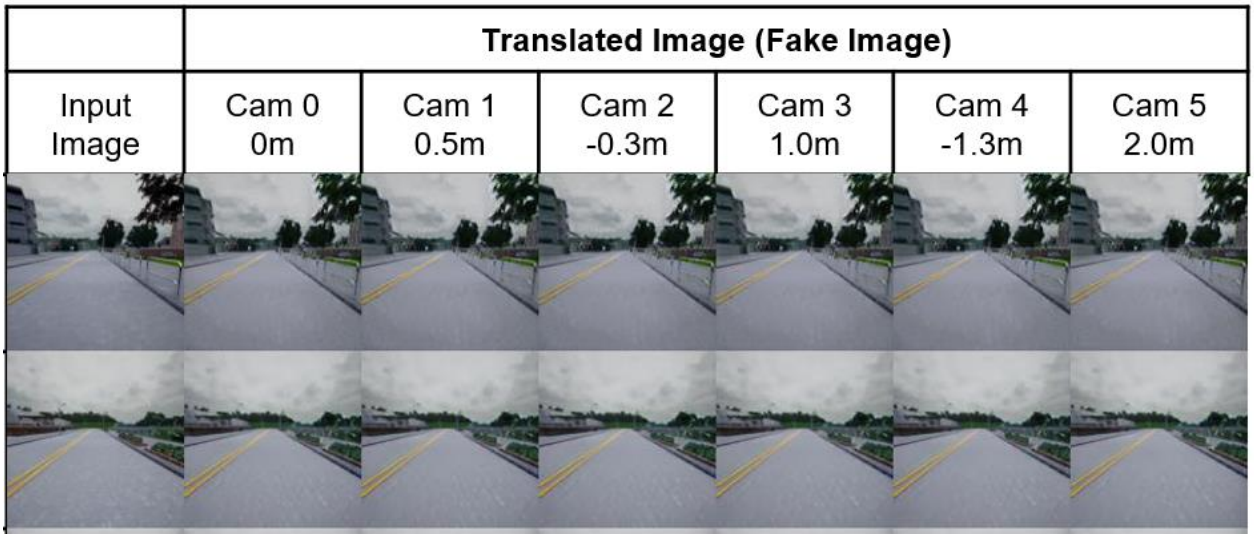


Figure 14: Only real/fake loss, the generator found a loophole by providing an output image identical to the input image.

3.2.3 Quantitative evaluation metric

Evaluating the performance of GANs is still open research. It is hard to define a quantitative metric that correlates with human perception. Especially in style translation, even humans may argue about the ground-truth. In our case of camera shifting, we already have an objective ground-truth which makes things a lot easier.

We tested four configurations for StarGAN tailored conditioning on the relative distance of the cameras:

- Config 1: With cycle loss
- Config 2: No cycle loss and classification loss for source image
- Config 3: No cycle loss and no classification loss
- Config 4: No cycle loss and classification loss for source image and target image

We then trained a model for 100k iterations for each of these configurations. To compare the results of these models with each other we used following three methods as image similarity metrics:

- MSE (Mean squared error): Is the most simple and naïve approach to quantify similarity between two images and is very susceptible to any structural transformations in the image. Lower MSE values indicate higher similarity between the images.
- SSIM (Structural similarity): Is the more common method to measure similarity between images which takes structure into account instead of just pixel to pixel difference. We used the SciKit-learn implementation of the method. Higher SSIM values indicate higher similarity between the images.
- Cosine distance of Feature Vector Extracted via VGG: A machine learning way to tell if images are similar, using a pretrained model (in our case VGG) extract feature vector for image and calculate the cosine distance. Lower Cosine distance values indicate higher similarity between the images.

Then using a common test set of 133 images we generate detailed statistics about the similarity of images of each method with the ground truth. Following table shows the average performance of each model on each evaluation criteria.

	VGG Cosine distance	SSIM	Mean Squared Error
Config 1	5.72	0.773	0.339
Config 2	6.12	0.829	0.246
Config 3	6.12	0.829	0.246
Config 4	5.58	0.856	0.265

Table 1: Comparison of different configurations

From the table above, if we ignore the naïve measure of MSE, both VGG Cosine Distance and SSIM declare config 4 as the winner. Which means config 4 (using no cycle loss & using classification loss for both source image and target image) provides with best reconstruction of image from a camera.

However, the above evaluation is based on when the camera for test also had a disjoint set of images in in training. Which arises the question about how our best model would perform if the camera at test was not a part of training at all?

To answer that, we trained another model based on config 4. This time we didn't include images from camera 2 during training. During testing, we predict the images of camera 2 and compare its result with our model from before, containing camera 2 in the training.

	Trained with Camera 2	Trained without Camera 2	Difference

VGG Cosine distance	5.58	6.10	1.11
SSIM score	0.856	0.791	0.0645

Table 2: Comparison of our config 4 model trained with and without camera 2 images and evaluate on camera 2.

As expected, by looking at the table above, we can see that the performance drops if we try to predict missing camera. However, with our synthetic dataset, can be argued that the loss is not substantial.

4 Conclusion

In this work, we first tested two different approaches, namely MUNIT and StarGAN, to do image translation of weather domains. While MUNIT didn't produced good results, the quality of StarGANs translation are convincing for both real and simulated data.

Afterwards, we extended StarGAN to work on continuous conditioning and improved the architecture to exploit the advantages of paired image data. We tested our approach on the task of camera shifting.

Reference

Choi, Y., Choi, M., Kim, M., Ha, J. W., Kim, S., & Choo, J. (2018). Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 8789-8797).

Huang, X., Liu, M. Y., Belongie, S., & Kautz, J. (2018). Multimodal unsupervised image-to-image translation. In Proceedings of the European Conference on Computer Vision (ECCV) (pp. 172-189).

Zhu, J. Y., Park, T., Isola, P., & Efros, A. A. (2017). Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision* (pp. 2223-2232).

DeepAI: https://deepai.org/machine-learning-model/image-similarity