Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Control Engineering and Information Technology

Label-consistent sim-to real image-transformation using neural networks

Scientific Students' Association Report

Author:

Solt Skribanek

Advisors:

dr. Márton Szemenyei
Róbert Moni

2021

Contents

Kivonat

Újabban nagy érdeklődés övezi az önvezető járműveket; az ipar szereplői is egyre több hangsúlyt fektetnek az ezzel kapcsolatos kutatásra és fejlesztésre. Az autonóm ágensek jellemzően mély neurális hálózatok, amelyek tanításához nagy mennyiségű felcímkézett adatra van szükség, ezek előállítása pedig komoly erőforrásokat igényel. Léteznek azonban szoftveres szimulátorok, amelyekben közvetlenül rendelkezésre állnak a tanító adatok, továbbá alkalmazásukkal ritkán előforduló vagy kockázatos szituációkra is felkészíthetjük az algoritmusainkat.

Komoly kérdést vet fel azonban, hogy a szimulátorban jól teljesítő ágens a való életben is tud-e működni. A jelenlegi state-of-the-art megoldások sajnos erre nem képesek, melynek oka, hogy a szimulátor által generált adatok nem elég valószerűek. Képek esetében ez azt jelenti, hogy rendkívül primitív és túlzottan homogén textúrákkal rendelkezik a szimulátor, így nem képes a valóság ábrázolására. Megoldást egy olyan képtranszformáció jelentene, amely a képen található objektumokat helyben hagyja, csak azok textúráit alakítja realisztikusabbá.

Habár erre az ún. domain transfer problémára számos megoldás született az elmúlt időben, ezek valamennyire megváltoztatják a kép szemantikus tartalmát is a textúra mellett, továbbá jobbára két valódi domain között működnek eredményesen. Ebben a dolgozatban újszerű megoldást adunk a problémára a saját Label-Consistent Swapping Autoencoder névre hallgató architektúránkkal, amely a Swapping Autoencoder továbbfejlesztett változata. Két új hibafüggvényt mutatunk be, amelyek arra hivatottak, hogy kikényszerítsék a szemantikus egyezést a bemenet és a kimenet között. Megmutatjuk, hogy ezek alkalmazásával a képtranszformáció még pontosabbá tehető.

State-of-the-art módszereket használva vetettük össze megoldásunkat korábbiakkal, valamint a céldomainen előre tanított szegmentáló hálózatot is igénybe vettünk a szemantikus egyezés vizsgálatához. Azt találtuk, hogy a kiegészített architektúra jobban megőrizte az objektumok térbeli helyzetét a textúracsere folyamán, így elmondhatjuk, hogy az újításaink beváltották a hozzájuk fűzött reményeinket.

Abstract

Interest in autonomous driving has grown tremendously in the last few years, with industry players also emphasizing the research and development in this field. Autonomous agents are typically deep neural networks, which require a large amount of expensive labeled training data. However, there are simulator softwares available in which labeled data is directly accessible. Using simulators in autonomous research comes with further advantages, for example algorithms can be prepared for dangerous or rare traffic situations at no additional cost.

However, the question is whether an agent trained in a simulator is able to operate in real world environments. Current state-of-the-art methods are not able to do so, because the simulated data are not realistic enough. In the field of vision, simulated images usually have very primitive and homogeneous textures, and thus they are not able to represent reality.

The solution to this problem could be an image-transformation that is able to convert the textures to be more realistic, while preserving the positions of the portrayed objects. There are numerous solutions for this so-called domain transfer problem, but they usually corrupt the semantic content of the image during the transformation in some way. They also tend to work well only on datasets based on real images. In this work, we propose a novel architecture called the Label-Consistent Swapping Autoencoder, which is an improvement to the Swapping Autoencoder. We introduce two new objectives to the neural network, whose goal is to enforce the semantic consistency between the input and output image. We show that using these, the style-transformation can achieve better results.

We compared our method with prior state-of-the-art works using a segmentation network pretrained on the target domain to study semantic consistency. We found that the improved architecture could better preserve the positions of the original objects, so we can determine whether our method reached its goal.

Chapter 1

Introduction

Autonomous driving has been enjoying an unbroken popularity in the last few years; more and more industry players in the automotive field are trying to get involved in the research and development of self-driving cars.

A key element of autonomous driving algorithms is the fast and reliable perception of the surroundings, which can provide the required information for the higher level decision-making. Although there are numerous other sensors used in modern self-driving development, visual information and camera-based systems play a major role in the detection.

Image-processing and decision-making methods nearly always use some sort of deep neural network, and the training of these networks requires a large amount of labeled training data, which can be prohibitively expensive. Software-simulators can, however, reduce the cost of the development, as training data with the corresponding labels are directly accessible in them. Using such simulators comes with further advantages: we can prepare the car for dangerous or extremely rare traffic situations at no additional cost.

Despite having such benefits, one can also face difficulties when trying to use a software-simulator like this. The data gained from the program is not realistic enough, which means an agent performing well in a simulated environment won't necessarily be suitable for driving on the street. Regarding images, simulated ones usually have very homogeneous and primitive textures, and thus they are not able to represent reality.

The solution to this problem could be an image-to-image transformation that is able to convert the textures to be more realistic, while preserving the positions of the portrayed objects. This way, we could benefit greatly from the simulator as

we could utilize the generated labels and also traing the agent using more realistic pictures.

Such image translation can be achieved via generative neural networks. There are many proposed solutions to this so-called domain transfer problem, but they mostly work only between two real datasets or corrupt the semantic content of the picture in some way. In this work, we added two new features for a state-of-the-art architecture called Swapping Autoencoder [25] that we found very promising for the texture-swapping task. Our new loss functions are based on the fact that we possess the semantic labels of the synthetized images and have also access to real labeled datasets. We can utilize these new information in a novel way to force the translation network to keep the semantic meaning of the picture during the transformation. We named these functions Inner Semantic Loss and Outer Semantic Loss.

The Inner Semantic Loss' responsibility is to constrain the autoencoder's latent representation to resemble the original semantic meaning of the picture. On the other hand, the Outer Semantic Loss examines the final translated image, and penalizes the generator for the semantic differences. With these two innovations, the full architecture is called Label-Consistent Swapping Autoencoder. We show that it can outperform the baseline Swapping Autoencoder using a semantic segmentation network pretrained on the target dataset. We also utilize modern metrics to compare translation results with real streetview images.

We organize our report in this way: the second chapter explains the basics of neural networks, and focuses directly on image-processing tasks. We will look at the semantic segmentation task since we uzilized its concepts in our innovations. We will also discuss the basics of generative models, and turn to the state-of-the-art approaches for the image-to-image translation task. Thereafter, in the third section, we will present our Label-Consistent Swapping Autoencoder, introducing the 2 new loss functions. We will also explain the implementation details of these innovations. The 4th chapter presents our experiments. In this section, we will shortly touch upon our synthetic dataset creation. After explaining the experimental setup using our novel architecture, we will demonstrate our results compared to a baseline model. In the last, 5th chapter we will draw conclusions and also talk about future improvements.

Our contributions are summarized as follows:

- We proposed a novel architecture called Label-Consistent Swapping Autoencoder by introducing two new objectives.

- We evaluated our method by a SOTA metric and a semantic segmentation network and found that our innovations reached their goal.
- We created an own dataset from a widely used simulator-software.

Chapter 2

Related Work

2.1 Neural networks

Neural networks (NNs) became popular in recent decades because of the massive growth in computational capacity and the growth in the amount and quality of the data that this field requires. Neural nets (in other words Deep Learning) is one of the forms of machine learning, an alternative of classical programming.

In the classical way, when someone writes a code or function, they shall know the relationship in detail between the input and output of the given function to be able to create it. Contrarily, machine learning, as its name tells, encourages the machine to learn the relationship or transformation between the input and output, typically by providing many coherent input-output pairs. This approach is very useful when the task cannot be mathematized in an explicit way, e.g. one can not tell exact formulas for the differentiating task between cats and dogs from pictures.

When using neural nets, the ability to learn diverse transformations is achieved by connecting very simple — but usually numerous — computational blocks. The name — neural network — comes from the analogy with the human brain, as researchers found that it is built from simple units — so-called neurons — with primitive duties, and the strength does not come from the complexity of the nodes, but from the large amount of the interconnections between these simple units. The nodes have many inputs, and their outputs are passed onto many other nodes.

In artificial NNs, the nodes typically apply a weighted summation on their inputs, followed by an non-linear function on the sum. Neurons are organized in layers, meaning that more nodes get the same inputs. In this way, the different neurons in the same layer provide the input for the next layer and so on. The final layer depends on the task, e.g. a binary classification problem implies a single scalar

as output, but when classifying into more classes, the final layer shall produce more numbers. These outputs are then considered as probabilities for the corresponding category.

NNs are usually trained by applying backpropagation [11], meaning that we are able to compute the gradients of the model's inner parameters (the weights and biases of the weighted summations) by comparing the model's prediction and the ground truth output for a given input. Then, we modify the parameters according to the computed gradient in small steps. To be able to do this, we should tell the network the expected output for each input. In supervised learning, we provide the data by labeling them in advance. In addition, we should provide a great quantity of labeled data in order to force the network to learn principal rules rather than becoming a look-up-table for the training data. Because of the small training steps and the big amount of data, a training takes a long time and also requires enormous computational capacity.

2.1.1 Image processing

When applying NNs to computer vision tasks, we must first consider some of the specialities of this field. Most importantly, images contain a relatively large amount of data to be processed: the digital representation of a single image with the current resulutions contains millions of pixels, not to mention the color channels. We can not use the so-called fully connected layers described above to weight all the pixels, because they would require too many parameters and so too large amount of memory.

From another point of view, applying fully connected layers would lose information about the contiguity: dense layers would flatten our 2D-shaped images, so that pixels above and below each other would not be neighbours.

However, using convolutional layers solves both difficulties. The convolution operation can be understood as a shared-weight matrix-multiplication: we need considerably fewer parameters, but still can create deep networks of these layers. 2D convolution also takes neighborhood into account, as the adjacent locations affect each other. A typical convolution layer has a shape of $H \times W \times C_{in} \times C_{out}$, where C_{in} and C_{out} are the numbers of the input and output channels. At the first layer, channels mean the color channels of the image, but deeper into the net, we refer to these as feature maps as they contain information about more and more complex features. The convolution layer can be understood as it has a 2D convolution kernel for all input channels for a given output channel, so there are ultimately $C_{in} \times C_{out}$ of

them. Networks applied to image processing tasks are therefore CNN-s [20], [12] — convolutional neural networks —, as most of the layers utilize convolution in some way.

Typically, the numbers of channels grow as we go deeper into the network as more complex features are discovered. In order to cut back memory-consumption, we reduce the spatial dimensions of the image (we call them feature maps after the first layer) by pooling operations or by strided convulutions. It also allows the convolutional kernel at the same size to have bigger field of view, in other words we increase the output stride.

The most widely used CNN is ResNet[13] or residual net. This network is composed of blocks; in every block there is a skip-connection from the input to the output of the given block in order to assure the gradient-flow. The name tells that the layers of the block should only learn the remaining difference beside the identical transformation.

2.2 Semantic segmentation

Semantic segmentation is a high-level image processing task. The goal is to classify each pixel of an image to a class, resulting in $N_{classes}$ binary masks *at the original resolution*. This requirement causes a difficulty: as we described above, basic CNN-s successively reduce the spatial dimensions of the feature maps as we approach the final output. It is not a problem for a simple image-classification task, but it is problematic when segmenting semantically, because we cannot restore the fine detailed information from the low-resolution feature maps.

As we need the feature masks in detailed resolution, after reducing the feature maps, we should grow them back to the original size. This can be achieved by learned or by fixed upscaling methods. An example for learned techniques is the transposed convolution, where the kernel's weights can change during training. Fixed upscaling practices include simple interpolations as nearest neighbor, bilinear or bicubic, and also a special one for neural nets that is called unpooling. With these techniques, we get into an encoder-decoder architecture.

However, by simply upscaling the low-resolution features either by learning or by fixed methods, we do not get the detailed output that we want. Therefore, another assumption is to make skip-connections between the shallow layers — that possess fine detailed informations — and the upscaled feature maps. This way, the fusion of high-level low-detail information and low-level high-detail features allows

the network to produce the expected outcome. It was the main idea of the first segmentation network, FCN [22]. The popular U-NET [26] improves the architecture further by making skip-connections on every scale.

The DeepLab [1, 2, 3, 4] networks constitute another popular and modern architecture family. They use a special block named ASPP (atrous spatial pyramid pooling), which performs several strided convolutions in paralell on the same input feature map, then fuses their outputs (Figure 2.1. Because of this, ASPP's output has a large field of view with little increase in the number of parameters. DeepLab also uses conditional random fields on the final output predictions. In this work, we utilize the newest member of the family, DeepLabV3+ [4] with ResNet [13] and MobileNet [16] backbones. Contrary to the robust and large ResNet architecture, MobileNet is a smaller classification network especially made for mobile applications.

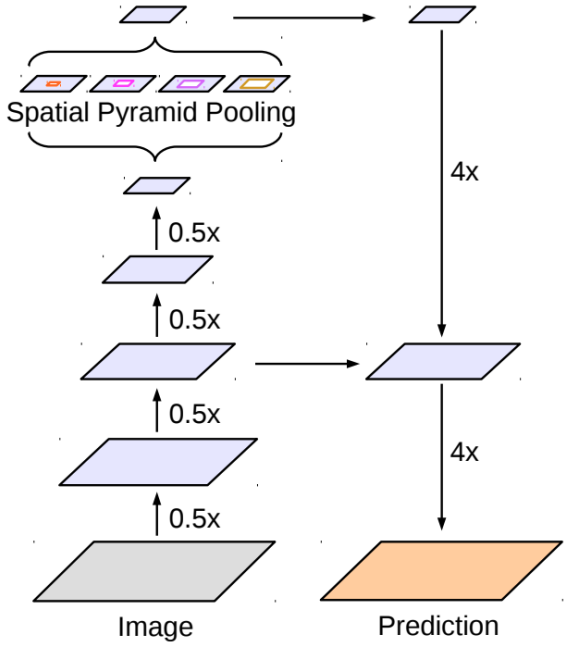

Figure 2.1. Schematic of the DeepLabV3+ encoder-decoder type architectecture with the ASPP module on top. Source: [4]

2.3 Generative models

Before we turn to the style-transfer problem and its proposed solutions, we need to touch some elementary building blocks of the generative neural networks. We use generative models for creating data like a large training dataset. To do this, we have to force the model to learn the underlying essence of the data to be able to generate

samples similar to them. If the training dataset contains much more samples than internal parameters the model has, this constrain is met.

2.3.1 Autoencoder

The classical autoencoder is an encoder-decoder structure that is able to produce a compact representation of the input in a latent space. It is important that the dimensionality of the latent space must be less than the input's, so the encoder is forced to find the best representation in the narrower space with the least information loss possible. The decoder's task is to reproduce the input from this compact latent representation as accurately as possible. Figure 2.2 shows us the simple architecture.

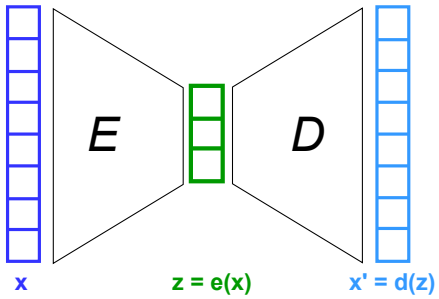

Figure 2.2. Sketch of the classical autoencoder

The training objective is to minimize the L1 or L2 (MSE) loss between the input and the output as follows:

$$\mathcal{L}_{AE} = ||x - x'||_{p\in\{1,2\}} = ||x - d(x)||_{p\in\{1,2\}} = ||x - d(e(x))||_{p\in\{1,2\}}$$

Using multi-layer neural networks is a great way to build both the encoder and the decoder.

Notbaly, there is a more modern architecture called Variational Autoencoder (VAE) that can be better utilized in image-generation tasks. VAE does not map its inputs into latent codes directly, rather it learns the distributions of the latent codes of the encoded training images. This way, the encoder provides information about the expected values and the covariance matrix of the latent parameters, so that the decoder can randomly sample from it, creating an image that differs from the training set.

2.3.2 GAN

The other elementary generative network is called GAN (Generative Adverserial Networks)[10]. Since its release, this idea has revolutionalized the field of AI-based image synthesis, along other data synthesis tasks. Most of SOTA methods utilize GANs in some way to create realistic images.

The basic setup of GANs is as follows: there is a single unlabeled dataset — a typical unsupervised setup — containing real images, and we want the network to be able to produce new images that are different but indistinguishable from the training images. It means that the network should learn the charactesistics of the training images to make synthesized images similar to them. Main idea of the GAN is that we achieve this by applying two adverserial neural networks: one is called generator, the other is called discriminator. The generator's job is to synthetize realistic (or at least indistinguishable) data from a randomly sampled noise-vector to fool the discriminator, while the discriminator's role is to distinguish the real datapoints from the synthetized ones. A simple sketch of the GAN can be observed on Figure 2.3.

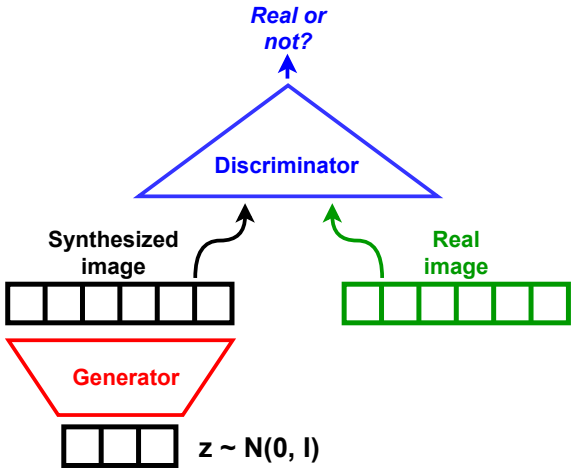

Figure 2.3. Scheme of a classical GAN architecture

Thus, discriminator is a simple binary classifier that aims to maximize its output in the case of real (training) image and minimize it if feeded with a synthetized image. The generator has an opposite goal: it aims to minimize the second term. The objective for both parts can be fromulated as a min-max game:

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

We can notice that the generator does not see training images directly, it can only draw conclusions from the discriminator's behavior. From a practical point of view it means that we train the generator by backpropagating through the whole discriminator network.

We train both parts simultaneously: first, we generate images utilizing the generator with freezed weights and compare them with randomly selected images from the training set. We refresh the weights of the discriminator, then we freeze them to train the generator. We generate images with it and discriminate them, then backpropagate through the whole network to refresh the generator's parameters.

However, GAN's training raises some problems. One of them is the so-called collapse during training: in many cases the discriminator defeats the generator at the beginning, and the generator cannot learn anymore. There are techniques to minimize this effect, for example the R1 regularization [23] that penalizes the discriminator's gradients on real data. Other difficulty is the training of the generator, because the gradients can vanish while flowing through the whole discriminator architecture. Novel GANs therefore use special activations in the discriminator to help the gradient's flow such as Leaky ReLU rather than the classical ReLU [19],[28].

2.4 Style transfer

Neural style transfer is a widely researched area, as generating photorealistic images based on existing ones is advantageous in many situations, such as image augmentation to increase the performance of CNNs used in computer vision [8], artistic stylization [31] or editing [25] of pictures or sim-to-real image-translation. We aim to do the latter: we want to reduce the domain-gap between real images and synthesized images from a simulator to aid the research on autonomous vehicles. In this section, we take a look at recent proposals that inspired our work.

2.4.1 CycleGAN

CycleGAN [31] by Zhu, Park et al. is a fundamental work in this topic. It was presented in 2017, and since then, it became one of the most cited publications in this field, and is considered as a baseline. Back then, there already existed some image-translation methods based on neural networks and GANs, but they were only able to solve the problem in a special case, when paired datasets were available for the training, i.e. a bijection between individual elements of the domains. CycleGAN was the first method that solved the image-translation without this proper "dictionary"

available, so it worked in an unsupervised manner. The only supervision is that one should separate the pictures into 2 folders — it is called domain-supervision. CycleGAN's main idea that there should be not only one but two transformation functions between domains X and Y. $G : X \to Y$ maps from X to Y, while $F : Y \to X$ does its opposite. The generators input is not noise in this case, rather an image from the source domain that should be transformed to the other. There are also two discriminators that discriminate if the given image comes from their respective domains. This objective function can be formulated as:

$$\mathcal{L}_{\text{GAN}}(G, D_Y, X, Y) = \mathbb{E}_{y \sim p_{\text{data}}(y)}[\log D_Y(y)] + \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log(1 - D_Y(G(x))].$$

between $G : X \to Y$ and its discriminator D_Y. CycleGAN is a symmetrical architecture, so F and D_X are similar to G and D_Y.

G and F trained along these principles can (at least in theory) create pictures that are indistinguishable from the target domain's images, but it does not guarantee the correlation of the input and the output images. The so-called cyclic consistency presented at CycleGAN provides an answer to this problem: it means that the two functions performed one after the other should restore the original image, so $F(G(x)) \approx x, \forall x \in X$ and similarly $G(F(x)) \approx y, \forall y \in Y$. This behavior is enforced by the *cycle consistency loss*:

$$\mathcal{L}_{\text{cyc}}(G, F) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\|F(G(x)) - x\|_1] + \mathbb{E}_{y \sim p_{\text{data}}(y)}[\|G(F(y)) - y\|_1]$$

The full objective of the architecture is a combination of $\mathcal{L}_{\text{cyc}}(G, F)$, $\mathcal{L}_{\text{GAN}}(G, D_Y)$ and $\mathcal{L}_{\text{GAN}}(F, D_X)$.

2.4.2 UNIT

UNIT[21] was published at the same time as CycleGAN by NVIDIA researchers Liu et al. They take a completely new perspective on the task: they assume that the images of the different domains can be mapped to a common latent space, meaning that semantically matching images' latent code is the same in this common latent space (Figure 2.4).

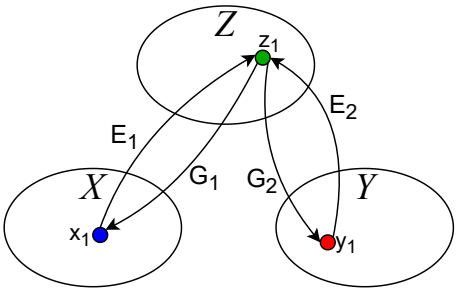

Figure 2.4. Shared latent space in the UNIT[21] method

The architecture consists of two combined VAE-GAN combinations. There is an encoder E, a generator G and a discriminator D for both domains. E and G form a VAE, G and D form a GAN in turn. The last layers of E_1 and E_2 and the first layers of G_1 and G_2 that carry the high-level, semantic information, and therefore their weights are shared to further constrain the common latent space. They also use a cycle-consistency loss-term: the two times translated image should represent the original. The VAEs and GANs are trained simultaneously with reconstruction, translation and cycle-consistency loss-terms.

2.4.3 Swapping Autoencoder

Zhu, Park et al. published another architecture called Swapping Autoencoder[25] in 2020 for texture swapping and photo editing. This method is fully unsupervised, meaning that there is not even a need for domain-supervision.

The main idea is related to the latent space once again. The core of the architecture is a special autoencoder (shown in Figure 2.5) that encodes the images into two latent components, where one is responsible for the structure, the other is for the texture of the image. The latent space is divided asymmetrically: the structure code z_s is a 3D tensor with spatial dimensions and a msall channel size, while the texture code z_t is a 1D vector. Our goal is to ensure that the reconstruction of swapped components results in a photorealistic image that represents the structure of the first input with the texture of the second input.

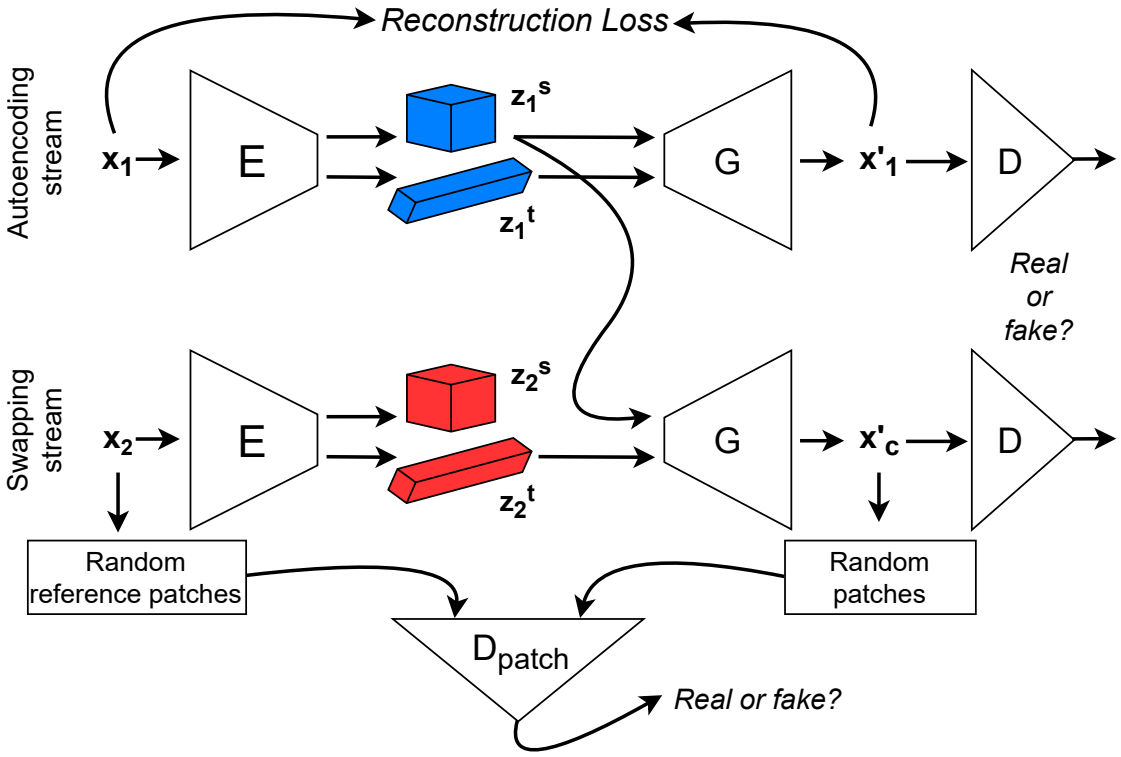

Figure 2.5. The Swapping Autoencoder

The reconstruction of the images must be accurate and realistic. The swapping autoencoder employs a classical L1 loss to ensure that the recombination is accurate:

$$\mathcal{L}_{\text{rec}}(E, G) = \mathbb{E}_{\mathbf{x}\sim \mathrm{X}} \left[\|\mathrm{x} - G(E(\mathrm{x}))\|_1 \right],$$

and it is assisted by a GAN-discriminator that is responsible for enforcing photorealistic reconstruction:

$$\mathcal{L}_{\text{GAN ,rec}}(E, G, D) = \mathbb{E}_{\mathbf{x}\sim \mathrm{X}}[-\log(D(G(E(\mathrm{x}))))].$$

They also enforce the decoder (or generator) to create realistic images from swapped latent codes that come from different pictures using a GAN-loss, termed:

$$\mathcal{L}_{\text{GAN,swap}}\left(E, G, D\right) = \mathbb{E}_{\mathbf{x}^1, \mathbf{x}^2 \sim \mathbf{X}, \mathbf{x}^1 \neq \mathbf{x}^2} \left[-\log\left(D\left(G\left(\mathbf{z}_s^1, \mathbf{z}_t^2\right)\right)\right)\right],$$

where $\mathbf{z}_s^1, \mathbf{z}_t^2$ are the structure code of image x^1 and the texture code of image x^2, respectively.

These constraints are enough to learn a factored representation, however, it is not necessarily true that z_t and z_s actually represent the texture and structure of the input. To address this, the authors encourage z_t to carry the texture information

by utilizing a so-called patch discriminator D_{patch}. Any images created with z_t should have the same texture, which means that small crops from them should be indistinguishable from small crops of the original image. The loss is formulated in this way:

$$\mathcal{L}_{\text{CooccurGAN}}\left(E, G, D_{\text{patch}}\right) =$$
$$\mathbb{E}_{\mathbf{x}^1,\mathbf{x}^2\sim\mathbf{X}}\left[-\log\left(D_{\text{patch}}\left(\text{crop}\left(G\left(\mathbf{z}_s^1, \mathbf{z}_t^2\right)\right), \text{crops}\left(\mathbf{x}^2\right)\right)\right)\right].$$

The size of the crops varies between $\frac{1}{4}$ and $\frac{1}{8}$ of the original image size.

Lastly, the final training objective is defined as a weighted combination of these losses:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rec}} + 0.5\mathcal{L}_{\text{GAN,rec}} + 0.5\mathcal{L}_{\text{GAN,swap}} + \mathcal{L}_{\text{CooccurGAN}}.$$

In each training interation, two images x^1 and x^2 are randomly sampled from X, then encoded and decoded, computing the $\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{GAN,rec}}$ losses on them individually. Finally, the $\mathcal{L}_{\text{GAN,swap}}$ and $\mathcal{L}_{\text{CooccurGAN}}$ losses are enforced on the hybrid image created from x^1 and x^2.

We shortly touch upon the implementation details of the encoder and decoder of the Swapping Autoencoder (shown in Figure 2.6), because we use this as a base for our Label-Consistent Swapping Autoencoder. We have to note that SAE borrows many ideas and elements from another state-of-the-art stylization method StyleGAN2 [19]: weight demodulation, antialiased bilinear down/upsampling, equalized learning rate, noise injection at every layer and the use of leaky ReLU. They also use StyleGAN2's discriminator almost directly, without minibatch-discrimination.

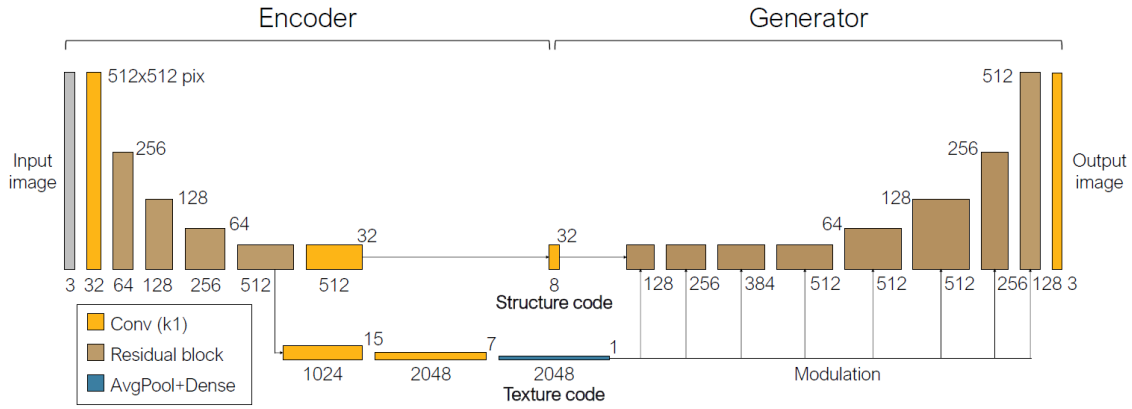

Figure 2.6. Encoder and generator architecture in the Swapping Autoencoder. Source: [25]

As described above, the encoder maps the input image into texture and structure codes. The common part consists of a convulutional layer and 4 downsampling residual blocks [13]. From this point, structure code is computed by two convolu-

tional layers, so that its shape is $H/16 \times W/16 \times 32$. For the texture code, the network branches off from the common point and adds 2 convolutional layers, followed by an average pooling layer and a dense layer. In order to lose positional information, the convolution layers in this branch use no padding and average pooling. The generator maps the codes back to an image, using the structure code in its main branch and injects the texture code's information with the weight modulation/demodulation layer from StyleGAN2 [19].

2.4.4 Style-transfer using semantic information

As previously stated, we aim to benefit from the extra information that we have, namely the semantic segmentation maps that are available in training time for both domains. We therefore examined GAN-based image generation methods that utilize semantic information.

The first group of related works used the segmentation maps directly. A fundamental work in this field, pix2pix [17] was published again by CycleGAN authors Zhu and Isola et al. It uses conditional GANs, meaning that both the generator and the discriminator have access to the labels of the given input. It also means that we have to provide semantic labels for every training image, so our dataset must be a so-called paired dataset. Pix2pix's generator is a classical encoder-decoder architecture made of residual blocks [13], where the input is the semantic map instead of noise. Pix2pixHD [27] extends this baseline by using multi-scale generators and discriminators and extra information on the input such as object boundaries. Pix2pixHD is able to generate visually appealing images at 2048×1024 resolution.

Other works, such as SPADE [24] by Park et al. do not feed the segmentation maps directly to the network. It samples noise according to the classical GAN, and uses a special normalization layer, which enables the injection of the semantic information to the generator at every scale. Although these networks can create visually appealing images from semantic maps allowing users to manipulate images on the semantic level, they require a paired dataset at training time and semantic maps in inference time.

Sem-GAN [5] and CyCADA [15] do not feed the segmentation maps to the network, they only use them to compute loss-terms. They both utilize the cycle-consistency loss, meaning that they both have two generators in both direcrions between domains. Both architectures use semantic loss that is computed from the difference of the ground-truth label and the segmented output of the translation. Sem-GAN is a symmetric architecture and they train its two segmentation networks

alongside the translation models. In CyCADA [15] the architecture is asymmetrical as only the source domain is labeled. There is a pretrained segmentation network used on the source domain, and they constrain that the image before and after the translation should represent the original segmentation information. The goal of this architecture is to learn a segmentation function on the target domain.

Chapter 3

Methodology

We picked the Swapping Autoencoder as our starting point. We also considered CycleGAN [31] and UNIT [21], but former experiments showed that the gap between our synthetic dataset and real images is too large for them, resulting only in color transformation. Swapping Autoencoder does a real style-conversion, but semantic consistency is weak. Therefore, we aimed on improving this architecture by adding more constraints.

We first modified the original architecture so it could only create sim-to-real hybrids, i.e. it could only mix structure codes extracted from synthetic images with texture codes extracted from real ones. Therfore, the swapping GAN-loss is modified in this way:

$$\mathcal{L}_{\text{GAN,swap}}\left(E, G, D\right) = \mathbb{E}_{\mathbf{x}^1 \sim \mathbf{X^{sim}}, \mathbf{x}^2 \sim \mathbf{X^{real}}} \left[-\log\left(D\left(G\left(\mathbf{z}_s^1, \mathbf{z}_t^2\right)\right)\right)\right],$$

where $\mathbf{X^{sim}}$ is the synthetic, $\mathbf{X^{real}}$ is the real dataset, and $\mathbf{z}_s^1, \mathbf{z}_t^2$ are the structure code of image $\mathbf{x^1}$ and the texture code of image $\mathbf{x^2}$, respectively. Similarly, the co-occurrence patch-discriminator's loss is also modified, as only sim-to-real hybrids are created:

$$\mathcal{L}_{\text{CooccurGAN}}\left(E, G, D_{\text{patch}}\right) =$$
$$\mathbb{E}_{\mathbf{x}^1 \sim \mathbf{X^{sim}}, \mathbf{x}^2 \sim \mathbf{X^{real}}} \left[-\log\left(D_{\text{patch}}\ \left(\text{crop}\left(G\left(\mathbf{z}_s^1, \mathbf{z}_t^2\right)\right), \text{crops}\left(\mathbf{x}^2\right)\right)\right)\right].$$

The reconstruction GAN-loss and the L1 loss are computed for both datasets equally. We refer to this model as our baseline.

As mentioned above, we want to utilize the semantic labels to further constrain the style-transfer. We introduce two novel objectives to the network: the Inner Semantic Loss and the Outer Semantic Loss.

3.1 Inner semantic loss

Swapping Autoencoder encodes the input images into structure and texture codes, where structure code is a 3D tensor with spatial dimensions. It should represent the semantic information of the image, as it should not carry any information about the texture (assuming disentanglement of structure and texture). However, only two facts guarantee that structure code actually represents the semantic meaning of the image: the 3D shape of the code and the D_{patch} discriminator.

We help the encoder learn a more appropriate disentanglement by training the structure code using the semantic labels. We assume that if the structure code truly represents the structure, then the ground-truth semantic label maps could be computed from the structure code (or the structure path of the encoder) with a function called $I : Z^s \to Y_{/4}$ where Z^s is the space of the structure codes, $Y_{/4}$ is the space of the labels, downsampled twice. Our inner semantic loss can be termed as:

$$\mathcal{L}_{\text{sem,in}}(E) = \mathbb{E}_{\mathbf{x}\sim\mathbf{X^{sim}}\cup\mathbf{X^{real}},\mathbf{y_{/4}}\sim\mathbf{Y_{/4}}}\left[\mathcal{NLL}\left(I\left(\mathbf{z_s}\right),\mathbf{y_{/4}}\right)\right],$$

where $\mathbf{x}$ and $\mathbf{y_{/4}}$ are corresponding image-label pairs and $\mathcal{NLL}$ stands for the negative log-likelohood function that penalizes the deviation of the prediction from the correct label at each position. In our work, we used its reduced form that takes the mean of the losses across the spatial dimensions.

With all these, the combination of E and I can be considered as an universal semantic segmentation network that can predict the semantic labels at a lower resolution for both $\mathbf{X^{sim}}$ and $\mathbf{X^{real}}$ datasets. We used a smaller resolution at this inner segmentation network, because even with learned upscaling, the network can not predict higher resolution segmentation maps without skip-connections from earlier layers with larger scales. Smaller resolution also increases computational efficiency.

3.1.1 Implementation of the inner semantic loss

We completed the encoder with a side-branch (named inner semantic branch) that branches off from the last convolutional layer just before the structure code. Because the spatial dimensions at this point are 1/16 of the original image size, the inner semantic branch is an upsampling network to reduce the gap in the scales between its output and the ground-truth semantic maps. Thus, it performs 2 learned upscalings, as it is a composition of transposed convolutional layer and classical convolutional layer from StyleGAN2 [19], repeated twice. The channel-size is halved

at every upscaling. At the end, there is another convolutional layer with a kernel size of 1 to make the final pixel-level predictions.

This way, the side-branch can be considered as the decoder part of a small-scale semantic segmentation network. We did not use skip-connections from shallower layers, because that part of the main branch contains informations about the texrue as well. The ouptuts of this part are then compared with the ground-truth label masks using the negative log-likelihood loss that penalizes if the network does not predict the correct class, for each pixel. Figure 3.1 shows this part of the architecture.

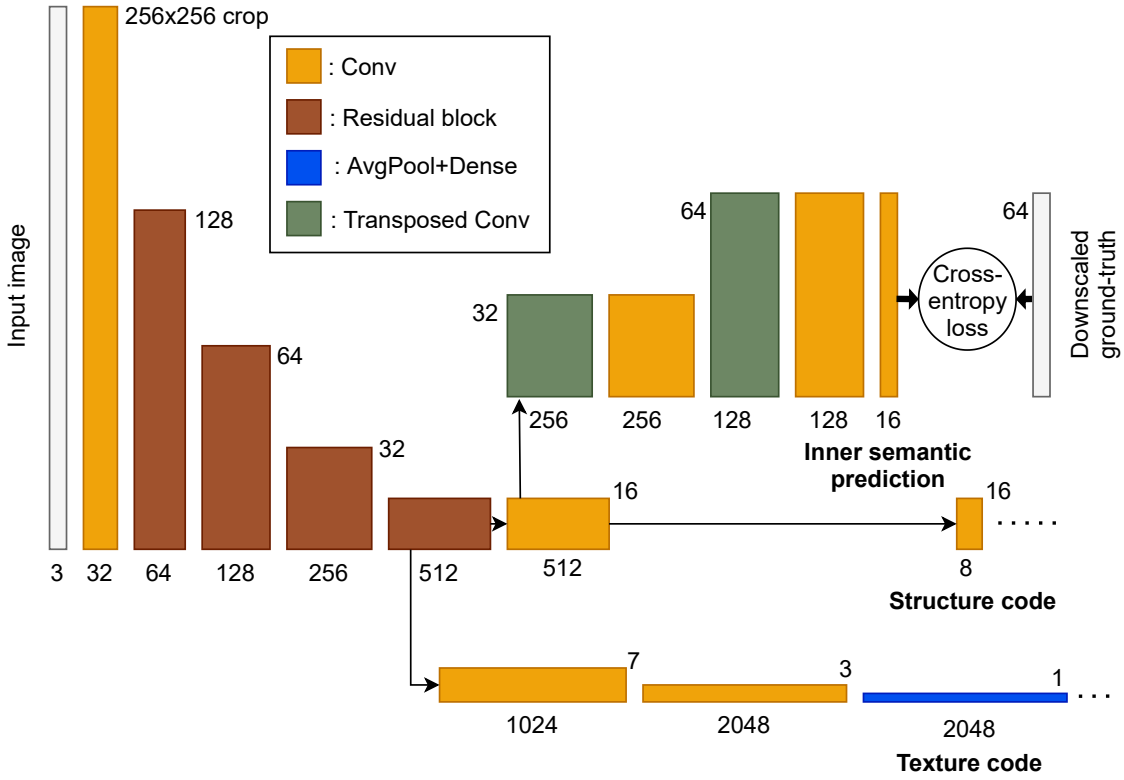

Figure 3.1. Completion of the encoder part of the Swapping Autoencoder with the inner semantic branch.

3.2 Outer semantic loss

We also constrainted our translation network from its outside: as we had access to the segmentation labels of the source dataset, we knew what the translated image should look like. We therefore prescribed that the result of the translation should have semantic content that corresponds to the original semgentation map. To be able to do this, we needed a function O that segments the translated images. If the style-transfer is performed correctly, then the textures of the translated image

correspond to the target dataset, meaning that a segmentation network pretrained on the target dataset with frozen weights is well-suited for the task.

We note that other works as Sem-GAN [5] also used a segmentation network similarly, but they trained their segmentation network simultaneously with the translation network. Also, at the beginning of the training, it could help the generator-network create objects that are similar to the real ones. The outer semantic loss can be formulated as:

$$\mathcal{L}_{\text{sem,out}}\left(E, G\right) =$$

$$\mathbb{E}_{\mathbf{x}^1 \sim \mathbf{X}^{\mathbf{sim}}, \mathbf{x}^2 \sim \mathbf{X}^{\mathbf{real}}, \mathbf{y} \sim \mathbf{Y}^{\mathbf{sim}}} \left[\mathcal{NLL}\left(O\left(G\left(\mathbf{z}_s^1, \mathbf{z}_t^2\right)\right), \mathbf{y}\right)\right],$$

where $\mathbf{y}$ is the label for $\mathbf{x}^1$ and $\mathbf{z}_s^1, \mathbf{z}_t^2$ are the structure code of image $\mathbf{x^1}$ and the texture code of image $\mathbf{x^2}$, respectively.

3.2.1 Implementation of the outer segmentation loss

We used a DeepLabV3+ [4] model with a MobileNet [16] backbone as the segmentation network O on the target dataset. We chose Mobilenet because of its small size: during training, we needed to backpropagate through the full network at each iteration, so we wanted to use as small network as possible to reduce the effect of vanishing gradients, and also increase computational efficiency.

3.3 Final objective

We combined our two new objectives with the former Swapping Autoencoder objectives, so our final objective is:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{rec}} + 0.5\mathcal{L}_{\text{GAN,rec}} + 0.5\mathcal{L}_{\text{GAN,swap}} + \mathcal{L}_{\text{CooccurGAN}} + \lambda_{in}\mathcal{L}_{\text{sem,in}} + \lambda_{out}\mathcal{L}_{\text{sem,out}}.$$

We kept the original weights of the SAE objectives, while λ_{in} and λ_{out} are hyperparameters.

In summary, we added two novel objectives to the Swapping Autoencoder [25], subsection 2.4.3 architecture, both penalizes deviation from semantic label maps: the inner semantic loss compares the structure codes with the downscaled labels, while outer semantic loss investigates a semantic segmentation of the translated image. The full architecture can be seen on Figure 3.2.

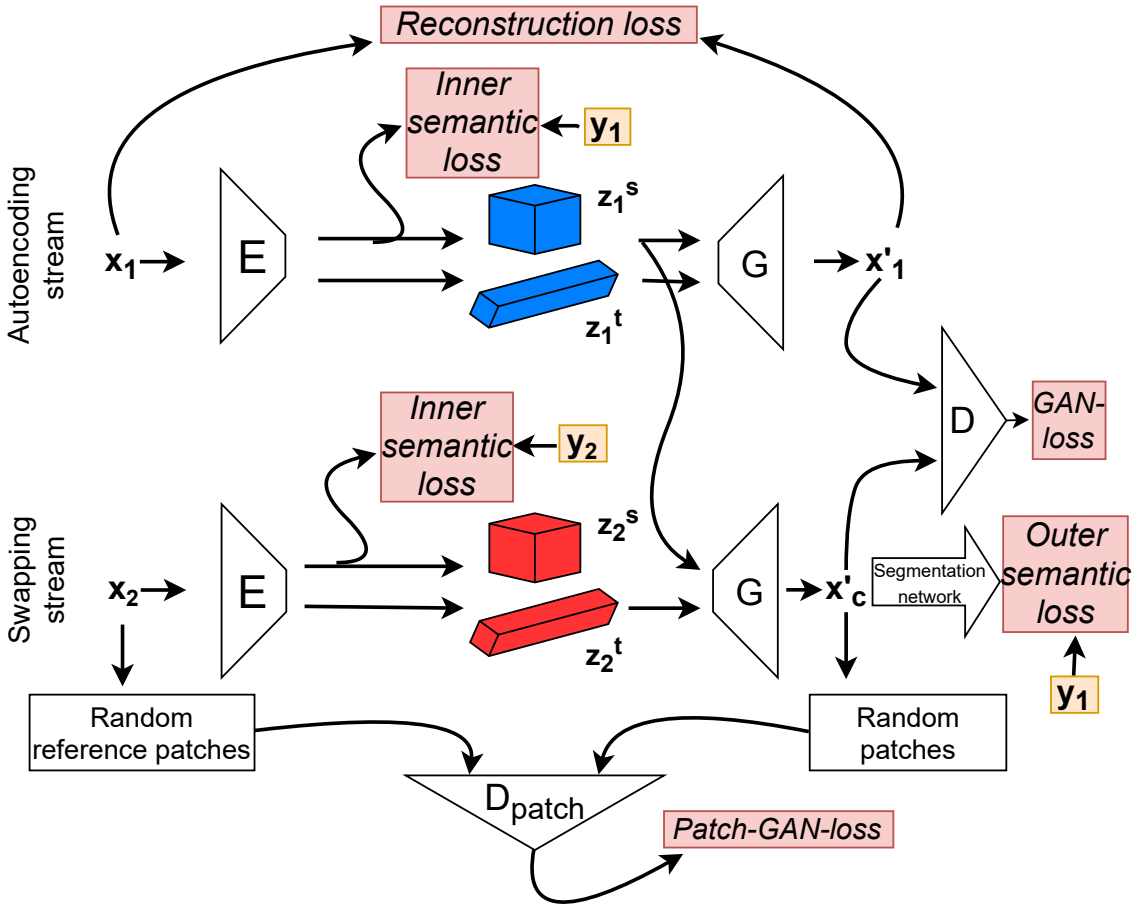

Figure 3.2. Full architecture of the Label-Consistent Swapping Autoencoder with its objectives

Chapter 4

Experiments

4.1 Datasets, data preparation

First, we needed a synthetic and a real dataset, both labeled. As real dataset we picked the widely-used CityScapes Dataset [6]. It contains 3475 finely labeled pictures at 2048×1024 resolution (2975 for training, 500 for validating) and another 1525 for test purposes.

To create our own synthetic set, we used the popular CARLA Simulator [7]. CARLA is an open-source and often updated software, specifically created to aid autonomous driving research. Although its visualization is very far from photorealism, it has numerous advantages compared to similar frameworks (i.e. computer games or expensive simulators) as it is completely free and provides a wide range of options for developing autonomous agents: it has several virtual sensors, while it also allows manipulation of the environment, the weather, the traffic situation and so on. In CARLA, we have the opportunity to create heterogeneous images as it has 8 different maps, ranging from rural environments to big cities.

We utilized CARLA's RGB and semantic camera sensors — the latter directly provides the semantic maps — and collected $20,000$ images with their corresponding labels. During data collection, we tried to create a set as similar as possible to CityScapes [6], so we varied the maps and weather accordingly, meaning that we collected more images from big cities and fewer from small towns, and allowed only dry weather without fog in daylight. Sample images can be seen in Figure 4.1.

Figure 4.1. Sample images from our CARLA Dataset

After the image generation, we also performed some filtering: we removed pictures that did not contain enough traffic actors (vehicles or pedestrians) based on the semantic maps, and we also deleted too similar pictures manually. At the end, we picked 5000 images (from which 2500 comes from Town10HD, the city with the most realistic textures) for training and another 100 for test.

We needed to modify the labels of the CityScapes Dataset [6] and our CARLA Dataset, because their labels were not consistent with each other. We first merged categories where it was needed. CARLA for example has a single *vehicle* category and *person* category, while CityScapes has separate classes for every vehicle-type *(car, truck, bus, etc.)* and for *pedestrian* and *rider*. On the other hand, we had to merge CARLA's *road* and *roadline* classes, because CityScapes did not have separate class for the roadlines. At the end, there remained 16 classes. After that, we downsampled the images and their labels to 512×256 resolution using Lanczos and nearest neighbor methods respectively in order to speed up the training.

In summary, we used two datasets for our training: the real set contains 3475, while the synthetic one contains 5000 labeled images, and the generated labels are consistent across the two datasets. The images and labels are at 512×256 resolution.

4.2 Setup

The original Swapping Autoencoder [25] worked on very large datasets, e.g. LSUN Church and Bedroom [29]: $\sim 126k$ and $\sim 3M$, FFHQ [18]: $\sim 70k$, in contrast, our dataset only contained $5000 + 3475 = 8475$ pictures. Due to the

relatively small size of our training dataset, we had to use some sort of augmentation. We used random 256×256 sized crops during training, since working on images of street scenes allows us this type of augmentation, as these pictures do not lose their meaning due to horizontal shifts. Fortunately, the fully convolutional architecture allows us to train on crops and test on the full images.

At each training iteration, we sampled N images from $\mathbf{X^{sim}}$ and N images from $\mathbf{X^{real}}$. We reconstructed $N/2$ real and $N/2$ synthetic images (chosen randomly from the inputs), and computed the reconstruction loss using these. We created N hybrid images, using all of the available structure and texture codes. The image discriminator's loss is computed on the $2N$ real, the N reconstructed and the N hybrid images, where both reconstructed and hybrid images are considered as fake. As for the patch-discriminator, we used its basic settings (8 crops for each image, averaged features for the reference image). Note that D_{patch} in our case works only on the target domain as it discriminates between sim-to-real hybrids and their textures that come entirely from the target domain.

For the other details, we also followed the basic setting of the Swapping Autoencoder [25], including the lazy R1 regularization with a weight of 10.0 for the image discriminator and 1.0 for the patch-discriminator, and the non-saturating GAN-loss. We used the basic optimizer ADAM with the default learning rate 0.002, $\beta_1 = 0.0$ and $\beta_2 = 0.99$.

We used an out-of-the-box DeepLabV3+ [4] model with Mobilenet [16] backbone as our outer segmentation network (`https://github.com/VainF/DeepLabV3Plus-Pytorch`), pretrained it on the resized and relabeled CityScapes [6] dataset with the original train-val split. The network reached 55% on the validation set, which is smaller than expected from DeepLabV3+ (claimed 72%). The significantly inferior performance is likely attributable to the heavy reduction in the input resolution. Table 4.1 shows the hyperparameters used for the training of this network. We used the maximum batch size that fitted into into the memory of a single 32GB Titan V100 GPU.

train	256×256 crops	iterations	$30k$
validation	whole image 512×256	learning rate	0.1
output stride	16	lr policy	polynomial
batch size	128	weight decay	0.0001

Table 4.1. Hyperparameters used to train our segmentation networks.

For the training of our Label-Consistent Swapping Autoencoder, we also used the maximum batch size that fitted into our memory: that is 8 when training with 256×256 crops, meaning that the network works with 16 images at the same time on our hardware. We set the maximum run-time to $5M$ iterations for all our experiments. Table 4.2 shows the important hyperparameters for our experiments.

batch size	8	total number of images	$5M$
crop size	256×256	learning rate	0.002
texture code dimension	2048	structure code's shape	$16 \times 16 \times 8$
λ_{GAN}	1.0	λ_{L1}	1.0
$\lambda_{patch-GAN}$	1.0	patch scale for D_{patch}	$1/4 - 1/8$

Table 4.2. The most important hyperparameters used in the training of the Label-Consistent Swapping Autoencoder

4.3 Results

We first trained our baseline model (i.e. $\lambda_{in} = \lambda_{out} = 0$), then trained another 5 models with different combinations of λ_{in} and λ_{out}. For evaluation, we used the 100 test images from our CARLA Dataset as structure images with 4 images from the CityScapes' [6] test set. With all the possible sim-to-real hybridizations, we get 400 result images.

As we wanted to increase semantic consistency, we used another segmentation network to evaluate it quantitatively. We picked a DeepLabV3+ [4] again, but with a more robust backbone ResNet50 [13] from the same repository (`https://github.com/VainF/DeepLabV3Plus-Pytorch`). This network has circa ten times as many parameters as the MobileNet [16] version. We pretrained it on our modified CityScapes Dataset [6]. For this training, we used the exact same hyperparameters as we used for the outer segmentation network. These can be seen on Table 4.1. We segmented all the 400 translated images, and compared it with the original CARLA images' ground-truth segmentation maps.

We used classical semantic segmentation metrics for quantitative investigation. The overall pixel accuracy metric is a ratio of the correctly predicted pixels. Mean pixel accuracy is computed by taking the mean of the pixel accuracies for each class. mIoU stands for mean intersection over union, it computes the intersection and the union of the correct and the predicted locations of a given class, then divides them, and takes the mean across the classes. The results can be seen on the Table 4.3.

We also employed the Fréchet inception distance (FID) [14] to measure the similarity of two image-datasets, as this metric is widely used to evaluate GAN-based image-generator networks. We computed the FID [14] metric between the 400 result images and the 5000 images of the CityScapes Dataset [6].

λ_{out}	λ_{in}	Overall Acc	Mean acc	mIoU	FID [14]
0.0	0.0	0.530	0.167	0.112	65.26
0.0	1.0	0.500	0.170	0.109	**60.13**
1.0	0.0	0.672	0.326	0.215	61.95
1.0	1.0	0.665	0.332	0.227	68.59
1.0	5.0	0.632	0.329	0.216	64.78
2.0	2.0	0.675	0.351	0.237	66.26
3.0	0.0	**0.692**	**0.382**	**0.254**	76.08

Table 4.3. Validation results using semantic segmentation metrics and FID [14] score. We **bold** the best results per column.

The table shows us that the use of outer segmentation loss highly increased the semantic consistency (doubled the mean pixel accuracy and the mIoU metrics) while the FID [14] metric did not change substantially ($\lambda_{out} < 3$ cases). It seems that the inner segmentation loss did not help the effectiveness of the network, as using only this loss lowered both pixel accuracy and mIoU compared to the baseline model. This likely means that the base encoder by itself could find a better representation (regarding the generation) than the strict semantic label maps. Notbaly, using the inner loss resulted in a somewhat sizeable reduction in the FID metric. The best performing model based on the semantic metrics is the one with $\lambda_{out} = 3$ and $\lambda_{in} = 0$, however, its Fréchet distance is significantly higher. The model with $\lambda_{out} = \lambda_{in} = 2$ can be considered as a best of both, because its semantic scores almost reach the best in this category, but there was no significant change in the FID score compared to the baseline model. Figure 4.2 reports some visual results of the translation network.

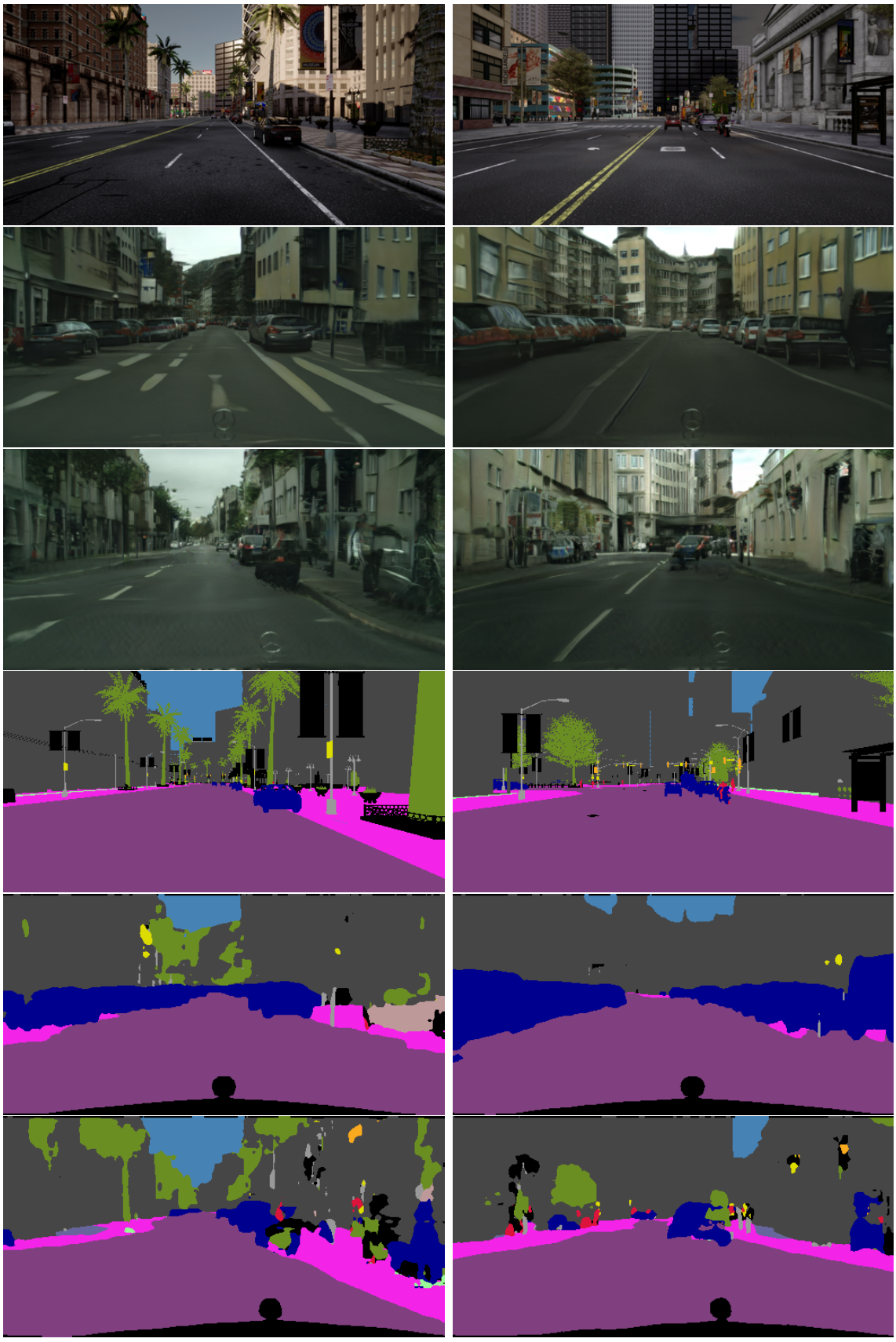

Figure 4.2. Visual results. From top to bottom: original picture from the CARLA set, transformed image with the baseline model and our best model ($\lambda_{out} = 3$), colorized ground-truth label, segmentation result of the baseline model's image and our model's image.

The above figure shows us the strength of our improvement: it prevents the translation from "hallucinating" cars next to the roads. CityScapes' [6] main advantage is that it is manually filtered and thus contains a large amount of traffic actors: cars and pedestrians. However, in our case, this appears as a weakness: there are too many cars in the CityScapes set, and therefore our GAN-based model collapses slightly: it can hardly imagine a road in the target domain without many cars parking aside. Our outer segmentation loss reduces this effect as it does not allow the translation network to park cars where there should not be any.

Figure 4.3 shows us another cases where our constraints resulted in more accurate translations as our model preserved the layout of the scene more accurately. The right side of the figure provides another demonstration of the weakness the CityScapes Dataset [6]: our real dataset contains only images captured in big cities, and there are very few images that portray the open sky, therefore our model places buildings or vegetation in place of the sky.

Figure 4.3. More visual results with another model ($\lambda_{in} = \lambda_{out} = 2$). From top to bottom: original image, translated image with the baseline and our model.

It is worth emphasizing, however, that our model is not perfect, as errors in the generated images sometimes still persist. For instance, our model ereases byciclists

and smaller objects on the road as seen on the left side of Figure 4.4, but still prevents adding unnecessary cars into the scene. There are also failure cases where our constraints ruin the translation: one example can be examined on the right side of Figure 4.4: the baseline places the car onto the right place while our model misses the object.

Figure 4.4. Another visual results

It can be said, however, that overall our model improves upon the baseline in far more cases than it fails, a claim evidenced by the significant improvement in the quantitative segmentation metrics. It is also worth emphasizing that we used a different segmentation architecture for training and evaluation to minimize the possibility that the generator simply learns to hide the semantic labels in the generated images.

Chapter 5

Conclusion

Sim-to-real image translation is a useful tool as it can aid the research of autonomous cars, as it can help creating more realistic videogames, or with its help we can make artistic images. In this work, we proposed a novel architecture for translating simulated images into a more realistic domain while also preserving its semantic content.

We investigated a state-of-the-art method called Swapping Autoencoder [25] made for texture swapping and proposed two new features to increase the semantic consistency of the translation. As an inner semantic loss, we added an semantic segmentation mini-branch into the middle of the architecture that constrainted that the structure code should match with the semantic labels of the image to be translated. We also utilized a semantic segmentation network pretrained on the target domain, and built it around the whole translation network to further enforce the semantic consistency of the output image.

We found that our inner semantic loss function had a negative effect on the consistency: this likely means that the Swapping Autoencoder with its original limitations is able to learn a more meaningful representation as structure code than the semantic label maps for the image-generation.

Our outer semantic loss highly increased the semantic consistency of the images as it doubled the mean pixel accuracy and the mIoU metric compared to the baseline. It is a notable outcome, but we believe it can be improved even further by additional considerations. With the unweighted crossentropy-loss, this objective forced the translation network to better preserve the layout of the scene, however, it attached greater importance to bigger objects like the road, trees and the sky, at vehicles' and pedestrians' expense. Therefore, we want to finetune the semantic constraints used in the outer semantic objective: we intend to differentiate between important and

not important categories. This way, we could punish the network more for replacing important objects as cars, pedestrians or traffic signs, and less for confusing not important categories as the sky or vegetation. We could weight our loss function based on classes, or instead of the cros-entropy loss that works pixel-wise, we could also use more sophisticated objectives, such as dice loss, a soft version of mIoU.

In our experience, the real dataset we used was not suitable for this task as it was small and too homogeneous as well. Our GAN learned the idiosyncracies of the target dataset — such as parking cars next to the roads, rich vegetation and big buildings that cover the sky etc. — along with the real-world textures, and this caused problems in the translation. In the future, we plan to use more datasets (e.g. KITTI [9], BDD [30]) merged with the CityScapes [6] to reduce the effect of this overfitting.

As for the translation network itself, we could try to condition the GAN with the semantic label maps, i.e. feeding the labels into the network at some point. In this case, however, one needs to be careful to avoid the generator network learning to hide the semantic information in the generated images.

Bibliography

[1] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[2] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2017.

[3] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[4] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[5] Anoop Cherian and Alan Sullivan. Sem-gan: Semantically-consistent image-to-image translation, 2018.

[6] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3213–3223, 2016.

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[8] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Gan-based synthetic medical image augmenta-

tion for increased cnn performance in liver lesion classification. *Neurocomputing*, 321:321–331, 2018.

[9] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 6.5. Back-Propagation and Other Differentiation Algorithms. MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*, chapter 9. Convolutional Networks. MIT Press, 2016. `http://www.deeplearningbook.org`.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[14] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. *Advances in neural information processing systems*, 30, 2017.

[15] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei Efros, and Trevor Darrell. Cycada: Cycle-consistent adversarial domain adaptation. In *International conference on machine learning*, pages 1989–1998. PMLR, 2018.

[16] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[18] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[19] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and improving the image quality of stylegan. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8110–8119, 2020.

[20] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[21] Ming-Yu Liu, Thomas Breuel, and Jan Kautz. Unsupervised image-to-image translation networks. March 2017.

[22] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[23] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for GANs do actually converge? In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3481–3490. PMLR, 10–15 Jul 2018. URL `https://proceedings.mlr.press/v80/mescheder18a.html`.

[24] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019.

[25] Taesung Park, Jun-Yan Zhu, Oliver Wang, Jingwan Lu, Eli Shechtman, Alexei A. Efros, and Richard Zhang. Swapping autoencoder for deep image manipulation. In *Advances in Neural Information Processing Systems*, 2020.

[26] 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.

[27] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation

with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[28] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[29] Fisher Yu, Ari Seff, Yinda Zhang, Shuran Song, Thomas Funkhouser, and Jianxiong Xiao. Lsun: Construction of a large-scale image dataset using deep learning with humans in the loop. *arXiv preprint arXiv:1506.03365*, 2015.

[30] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2636–2645, 2020.

[31] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, 2017.