UNIVERSITÀ DEGLI STUDI DI TRENTO

DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE
ICT International Doctoral School

Exploring Deep generative models for Structured Object Generation and Complex Scenes Manipulation

Pierfrancesco Ardino

Advisor

Dr. Bruno Lepri

Fondazione Bruno Kessler

Co-Advisor

Prof. Elisa Ricci

Università degli Studi di Trento

January 2023

Abstract

The availability of powerful GPUs and the consequent development of deep neural networks, have brought remarkable results in videogame levels generation, image-to-image translation , video-to-video translation, image inpainting and video generation. Nonetheless, in conditional or constrained settings, unconditioned generative models still suffer because they have little to none control over the generated output. This leads to problems in some scenarios, such as structured objects generation or multimedia manipulation. In the manner, unconstrained GANs fail to generate objects that must satisfy hard constraints (e.g., molecules must be chemically valid or game levels must be playable). In the latter, the manipulation of complex scenes is a challenging and unsolved task, since these scenes are composed of objects and background of different classes. In this thesis , we focus on these two scenarios and propose different techniques to improve deep generative models. First, we introduce Constrained Adversarial Networks (CANs), an extension of GANs in which the constraints are embedded into the model during training. Then we focus on developing novel deep learning models to alter complex urban scenes. In particular, we aim to alter the scene by: i) studying how to better leverage the semantic and instance segmentation to model its content and structure; ii) modifying, inserting and/or removing specific object instances coherently to its semantic; iii) generating coherent and realistic videos where users can alter the object's position.

Keywords

[Generative adversarial networks (GANs), Structured objects generation, Image manipulation, Video generation]

Contents

1 Introduction **1**

1.1 Conditional Deep Generative Models 1

1.1.1 Contributions 4

1.1.2 Outline 5

2 Constrained Adversarial Networks **7**

2.1 Related Work 8

2.2 Unconstrained GANs 9

2.3 Generating Structures with CANs 10

2.3.1 Limitations of GANs 11

2.3.2 Constrained Adversarial Networks 11

2.3.3 Conditional CANs 15

2.4 Experiments 16

2.4.1 Super Mario Bros level generation 17

2.4.2 Molecule generation 21

2.5 Conclusion 23

3 Semantic-Guided Image Inpainting **25**

3.1 Related Work 28

3.2 Approach 31

3.2.1 Learning to generate instances 32

3.2.2 Learning to inpaint 33

3.3 Experiments 37

3.3.1 Baselines 38

3.3.2 Experimental settings 38

3.3.3 Evaluation 39

3.4 Results 40

3.5 Conclusion 45

4 Controllable Video Generation **47**

4.1 Related Works 50

4.2 Click to Move framework 53

4.2.1 Object motion estimation with GCNs 54

4.2.2 Motion encoding 57

4.2.3 Generation module and training objectives 58

4.3 Experiments 60

4.3.1 Ablation Study 62

4.3.2 Comparison with State-of-the-art 63

4.4 Conclusion 67

5 Conclusion and Future Work **69**

A Appendix to Chapter 2 **73**

A.1 Implementation details 73

A.1.1 Super Mario Bros Level Generation 73

A.1.2 Molecule Generation 76

B Appendix to Chapter 3 **79**

B.1 Implementation Details 79

B.2 Dataset details 82

B.3 Use cases 82

B.4 Additional results 83

B.5 Edges contours 85

B.6 Additional ablations 86

C Appendix to Chapter 4 **91**

C.1 Network architectures 91

C.1.1 Convergence issue 91

C.2 Implementation. 92

C.3 Additional qualitative results 93

Bibliography **93**

List of Tables

2.1 Comparison between GAN and CAN on SMB level generation with pipes. The 7 maps containing pipes are *mario-1-1*, *mario-2-1*, *mario-3-1*, *mario-4-1*, *mario-4-2*, *mario-6-2* and *mario-8-1*, for a total of 1,404 training samples. Results report validity, average number of pipe tiles per level, L1 norm on the difference between each pair of levels in the generated batch and training time. Inference is real-time (< 40 ms) for both architectures. . 20

2.2 Results on the generation of *playable* SMB level. Levels *mario-1-3* (123 training samples) and *mario-3-3* (122 training samples) were chosen due to their high solving complexity. Results compare a baseline GAN, a GAN combined with CMA-ES and a CAN. Validity is defined as the ability of the A* agent to complete the level. Note that inference time for GAN and CAN is measured in milliseconds while time for GAN + CMA-ES is in minutes. 21

2.3 Results of using the semantic loss on the MolGAN architecture. The diversity score is obtained by comparing sub-structures of generated samples against a random subset of the dataset. A lower score indicates a higher amount of repetitions between the generated samples and the dataset. The first row refers to the results reported in the MolGAN paper. 23

3.1 Quantitative results for our model and the baselines. We evaluate the models through image quality (PSNR and FID) and accuracy of instance insertion (F1). 40

3.2 Ablation study of the architecture and input quality. 44

4.1 Ablation study results on Cityscapes. 62

4.2 Quantitative comparison in the *Oracle* and *Custom* setting. N is the number of user-controlled objects. $N = 1$ selects one object at random 63

A.1 Super Mario Bros Level Generation network architecture. . . 74

A.2 Reachability Network architecture. 76

B.1 Network architecture. We use the following notation: Z: the dimension of attribute vetor, N_C: the number of the class instance, θ: the number of location parameters of the affine transformation, K: kernel size, S: stride size, P: padding size, CONV: a convolutional layer, DECONV: a deconvolutional layer, FC: fully connected layer, IN: Instance Normalization . 80

B.2 Network architecture. We use the following notation: $\boldsymbol{m}$: the dimension of the mask, $\mathcal{C}$: the semantic classes, $\hat{\mathbf{m}}_s$: the dimension of the instance, K: kernel size, S: stride size, P: padding size, D: dilation factor, CONV: a convolutional layer, IN: Instance Normalization, SN: Spectral Normalization . . . 81

B.3 Quantitative results for our model and the baselines for the use cases of precise removal and maks insertion. We evaluate all the networks in two experimental settings and through image quality (PSNR and FID). 84

B.4 Ablation study of the components of the SPADE block. . . . 87

C.1 Network architecture. We use the following notation: Z: the dimension of motion vector, K: kernel size, S: stride size, P: padding size, CONV: a convolutional layer, UP: upsample, 3D-CONV: 3D convolutional layer, BN: Batch Normalization, SKIP: skip connection 92

List of Figures

2.1 Left: fuzzy logic encoding (using the Łukasiewicz T-norm) of $x \oplus y$ in CNF format as a function of $P(x = 1)$ and $P(y = 1)$. Middle: encoding of DNF XOR. Right: SL of either encoding. 13

2.2 Examples of SMB levels generated by GAN and CAN. Left: generating levels containing pipes; right: generating reachable levels. For each of the two settings we report prototypical examples of levels generated by GAN (first and third picture) and CAN (second and fourth picture). Notice how all pipes generated by CAN are valid, contrarily to what happens for GAN, and that the GAN generates a level that is not playable (because of the big jump at the start of the map). 19

3.1 Our holistic model can be applied in a wide range of manipulations in complex scenes. At inference time users can remove objects by either precisely indicating a mask or an entire area. Moreover, they can insert new objects (e.g. cars, pedestrians) by randomly generating them or by feeding a segmentation mask as input. 26

3.2 (left) Our model synthetizes a new image ($\boldsymbol{x}_{\text{FILLED}}$) and its segmentation ($\boldsymbol{s}_{\text{FILLED}}$) from an incomplete image ($\boldsymbol{x}_{\text{BLANKED}}$), its segmentation map ($\boldsymbol{s}_{\text{BLANKED}}$) and an optional instance, which might be sampled from a latent distribution ($\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$) or fed in input. The two discriminators encourage the generation of samples that resemble the real data distribution. (right) Our decoder block in the generator is based on SPADE [86]. We propose a modification to use the predicted segmentation map instead of the ground truth segmentation and adopt a multi-scale segmentation loss to better learn it. 30

3.3 Qualitative evaluation on the task of object insertion and inpainting. The first two rows show results on the Cityscape dataset while the last two rows show the Indian Driving results. We show the results on the same multi-domain model conditioned on two types of object insertion: cars and pedestrians. 41

3.4 Qualitative evaluation on the task of inpainting without object insertion. The first row shows results on the Cityscape dataset, while the last row shows the Indian Driving results. Zoom in for better details. 43

4.1 Illustration of the video generation process of Click to Move (C2M): 1) the user selects the objects in a scene and specify their movements. 2) Our network models the interactions between *all* objects through the GCN and 3) predicts their displacement. 4) The network produces a realistic and temporally consistent video. 48

4.2 Our network is composed of three modules, namely (i) Appearance encoding, (ii) Motion encoding, and (iii) Generation module. The Appearance Encoding focuses on learning the visual appearance from $\mathbf{X}_0$. The Motion Encoding models the interactions between the objects, predicts their displacement, encodes the motion, and generates the optical flow and occlusion mask for the Generation Module, which focuses on generating temporal consistent and realistic videos. On the right, we show our GCN module to model objects' interactions. 52

4.3 Qualitative comparison in the *Custom* setting on the Cityscapes dataset with ground truth reference. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details. 64

4.4 Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the Cityscapes dataset [14] with ground truth reference. On first three columns, we move the pedestrian near the semaphore to left. On the last three columns we move car crossing the street. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details. 65

A.1 A decomposed pipe 74

A.2 Given some generated levels (first column), this picture compares real reachability maps computed by the A^* agent (second column) and approximated by $\hat{P}_R$ (third column). 78

B.1 Qualitative evaluation on the task of object precise removal. The first two rows show results on the Cityscape dataset while the last two rows show the Indian Driving results. We show two types of object removal: cars and pedestrians. 83

B.2 Qualitative evaluation on the task of "mask insertion" object reconstruction. The first two rows show results on the Cityscape dataset while the last two rows show the Indian Driving results. We show two types of object insertion: cars and pedestrians. . 84

B.3 Additional results on Cityscapes reconstruct experimental setting. 85

B.4 Additional results on Indian Driving Dataset reconstruct experimental setting. 86

B.5 Additional results on Cityscapes Dataset insert & reconstruct experimental setting. 87

B.6 Additional results on Indian Driving Dataset insert & reconstruct experimental setting. 88

B.7 Results of [80] on the Cityscapes dataset. 89

B.8 Qualitative evaluation on the ablation study with and without $\mathcal{L}_{\text{style}}$. Removing $\mathcal{L}_{\text{style}}$ the network generate "checkerboard" artifacts. Zoom in for better details. 89

C.1 Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the Cityscapes dataset [14] with ground truth reference. On first three columns, we move the pedestrian near the semaphore to left. On the last three columns we move car crossing the street. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details. 94

C.2 Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the Cityscapes dataset [14] with ground truth reference. On first three columns, we move the pedestrian near the semaphore to left. On the last three columns we move car crossing the street. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details. 95

C.3 Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the Cityscapes dataset [14] with ground truth reference. On first three columns, we move the pedestrian near the semaphore to left. On the last three columns we move car crossing the street. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details. 96

C.4 Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the KITTI360 dataset [137]. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details. 97

C.5 Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the KITTI360 dataset [137]. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details. 98

C.6 Results of the ablation test on the Cityscapes dataset [14]. . . 99

Publications

This thesis consists of the following publications:

- Chapter 2:
  - Luca Di Liello*, **Pierfrancesco Ardino***, Jacopo Gobbi* (equal contribution), Paolo Morettin, Stefano Teso, and Andrea Passerini. Efficient generation of structured objects with constrained adversarial networks. *Advances in neural information processing systems, 33*, 2020.

- Chapter 3:
  - **Pierfrancesco Ardino**, Yahui Liu, Elisa Ricci, Bruno Lepri, and Marco De Nadai. Semantic-Guided Inpainting Network for Complex Urban Scenes Manipulation. *Proceedings of the International Conference on Pattern Recognition (ICPR)*, 2020.

- Chapter 4:
  - **Pierfrancesco Ardino**, Marco De Nadai, Bruno Lepri, Elisa Ricci, Stéphane Lathuilière. Click To Move: Controlling Video Generation With Sparse Motion. *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021

Chapter 1

Introduction

1.1 Conditional Deep Generative Models

In recent years, the availability of more powerful GPUs and deep neural networks have allowed deep generative models to show promising results in image-to-image translation [47, 70, 71], video-to-video translation [130], image inpainting [110, 143, 144], and video generation [85]. Nonetheless, unconditioned generative models have little to no control over the generated output. This leads to problems in some scenarios, such as structured object generation. In this case, Generative Adversarial Networks (GANs) [31] alone struggle to generate objects that have to satisfy hard constraints, like drug molecules generation, in which all molecules must be chemically valid. Another example is game levels generation, where we need to ensure that the generated levels are playable. The issue is that these requirements are usually difficult to acquire from examples alone, especially if the data are noisy. Existing works rely either on ad-hoc architectures [18] or use post-processing techniques that affect the inference time [117, 127]. For example, works based on probabilistic circuits [88, 57] generate structured objects but their inference grows linearly with the size of the circuits. As can be imagined, this could lead to problems if the constraints are complex. Other approaches based on reinforcement learning [18, 42] have tackled the problem of efficient sampling by incorporating a constrained learning component into the training procedure, and discarding the constrained part during the inference. Thus, in order to solve these tasks, it is necessary to develop a model that encodes constraints without affecting the time and the complexity of the sampling procedure.

Another task where unconstrained or unconditioned generative model struggle is multimedia manipulation. Multimedia manipulation refers to the process of changing the content or the style of videos and images with respect to the user needs. The possibility of changing the aspect of a scene is relevant for a large number of real-world applications including data augmentation [105], photo editing [68], and Augmented Reality (AR) [28]. Nonetheless, the manipulation of complex scenes is a challenging and unsolved task, since these scenes are indeed composed of objects and background of different classes. Moreover, these objects can be cluttered, occluded, or can appear during a sequence (in case of video sequence), making it difficult to apply conventional manipulation techniques.

In image editing, existing works focus either on removing or inserting an object. Song et al. [110] insert new objects by asking the user to draw precise segmentation pixels, while Hong et al. [40] only requires a bounding box, which is simpler compared to drawing segmentation pixels but lacks control. Instead, Lee et al. in [62] learn to place an object in plausible locations and with plausible shapes. The task of object removal has been addressed by image inpainting models that reconstruct the image from contextual pixels [144, 143]. Thus, the choice is between models that insert objects in controlled settings and models that inpaint corrupted areas. Our goal is exploring a novel framework in order to unify these two tasks.

Extending image manipulation models to videos has proved to be a challenging task. Video generation models have the additional task to model the time dimension, and provide motion and temporal consistency of edited objects. Initial attempts in video generation with deep generative models were based on generative adversarial networks (GANs). Videos can be generated starting from random noise [119, 96] in an unconditional setting or conditioned with other type of data such as images or semantic maps [85]. In the former users have usually little control on latent vector and it is challenging to modify some part of the scene. Indeed, it is difficult from a user perspective to change the content of the video or the position of some objects by only altering latent codes. In this direction, Wang et al. [130] have proposed a video-to-video translation model that takes as input a sequence of semantic maps or sketches and translate them to a video. Later, Pan et al. in [85] have proposed a conditional generation model that takes as input the first frame along its semantic representation and generates a sequence conditioned on it. However, the former requires the user to draw a sequence of sketches, which can be difficult and problematic in terms of motion consistency. In the latter, the user has little to no control on the motion of the scene. And, especially, on the objects in the scene.

In this thesis, we investigate how we can put humans in the loop and how we can improve the object interactions in the model. In general, three main topics are discussed in this thesis:

- **Constrained generation**. We propose a novel extension of GANs in which the constraints are embedded into the model during training.
- **Image manipulation**. Inspired by [40, 110, 131], we propose a model able to insert and remove objects in the scene, coherently with the rest of the objects in a single pass.
- **Controlled video generation**. Following the work of [85] on video generation and Casas et al. [5] on object interactions, we aim at developing

a framework where the user can control the motion of objects through mouse clicks. Moreover, the adoption of a Graph Neural Network (GCN) will model object interactions in the video.

1.1.1 Contributions

In Chapter 2, we tackle the problem of structured objects generation by introducing Constrained Adversarial Networks (CANs). CANs extend unconstrained GANs by introducing the embedded constraints into the generator. During the training, the generator is penalized whenever it outputs invalid structures. The penalty term is implemented using the semantic loss (SL) [139], which turns constraints into a differentiable loss function implemented as an arithmetic circuit. In contrast to other generative models, CANs support efficient inference of valid structures and allows to turn on and off the learned constraints at inference time. Moreover, we show how CANs can handle complex constraints that would be intractable to encode with propositional logic.

In Chapter 3, we propose a new deep learning method in the field of image manipulation. Inspired by works on image inpainting [110] and object insertion [40, 62], the proposed method leverages semantic segmentation to model the content and structure of the image, and learns the best shape and location of the object to insert. Our model gives the user the possibility to either draw an object or to let the network decide its shape and position inside the missing area. In this case, the generation of a new object is done using a Variational Auto Encoder (VAE) [54]. Then we use a one-stage architecture to insert the encoded object instances and inpaint the remaining part of the missing image using segmentation guidance. Our experiments on urban scenes datasets show that our proposed approach successfully addresses the problem of image inpaiting and object insertion/removal.

In Chapter 4 we continue to tackle the problem of multimedia manipulation but in the video generation scenario. Inspired by works on video generation [85, 103] and objects interaction in urban scenarios [5], we present a novel framework that generates video sequences with object motion guidance. This framework is composed of two main parts: (i) a Graph Neural Network that models the interactions between the objects in the scene and learn a distribution in order to sample new object trajectories, and (ii) a Deep Generative Model based on [85, 107] that takes as input an image, its corresponding segmentation map and the trajectories generated by the GNN and generates a video sequence conditioned on the trajectories. Given the conditioned nature of the framework, the user will have the possibility to decide the trajectories of some of the objects, while the GNN will take care of the other ones.

1.1.2 Outline

This thesis is organized in three chapters. In Chapter 2 we observe that Generative Adversarial Networks (GANs) [31] struggle in generating objects that satisfy hard structural constraint. Here, we propose Constrained Adversarial Networks (CANs), an extension of GANs in which the constraints are embedded into the model during training. In Chapter 3 we focus on controlling the manipulation of images of challenging scenarios, such as urban scenes. Indeed, urban scenes usually contain multiple semantics and objects, which can be frequently cluttered or ambiguous, thus hampering the performance of standard manipulation models. Here, we propose a novel deep learning model to alter a scene by removing a user specified portion of the image and coherently inserting a new object (e.g., a car or a pedestrian) in that scene. Chapter 4 analyzes how hard it is to generated coherent videos of complex scenes. To tackle this task, we introduce a framework for video generation where the user can control the motion of the synthesized video through mouse clicks specifying simple object trajectories of the key objects in the scene. Finally, we conclude in Chapter 5 where we also explore potential new directions.

Chapter 2

Constrained Adversarial Networks

Many key applications require to generate objects that satisfy hard structural constraints, like drug molecules, which must be chemically valid, and game levels, which must be playable. Despite their impressive success [53, 147, 152], Generative Adversarial Networks (GANs) [31] struggle in these applications. The reason is that data alone are often insufficient to capture the structural constraints (especially if noisy) and convey them to the model.

As a remedy, we derive Constrained Adversarial Networks (CANs), which extend GANs to generating valid structures with high probabilty. Given a set of arbitrary discrete constraints, CANs achieve this by penalizing the generator for allocating mass to invalid objects during training. The penalty term is implemented using the semantic loss (SL) [139], which turns the discrete constraints into a differentiable loss function implemented as an arithmetic circuit (i.e., a polynomial). The SL is probabilistically sound, can be evaluated exactly, and supports end-to-end training. Importantly, the polynomial – which can be quite large, depending on the complexity of the constraints – can be thrown away after training. In addition, CANs handle complex constraints, like reachability on graphs, by first embedding the candidate configurations in a space in which the constraints can be encoded compactly, and then applying the SL to the embeddings.

Since the constraints are embedded directly into the generator, high-quality structures can be sampled efficiently (in time practically independent of the complexity of the constraints) with a simple forward pass on the generator, as in regular GANs.[1] No costly sampling or optimization steps are needed. We additionally show how to equip CANs with the ability to switch constraints on and off dynamically during inference, at no run-time cost.

Overall, the main contributions of our work are as follows:

- CANs, an extension of GANs in which the generator is encouraged at training time to generate valid structures and support efficient sampling,
- native support for intractably complex constraints,
- conditional CANs, an effective solution for dynamically turning on and off the constraints at inference time,
- a thorough empirical study on real-world data showing that CANs generate structures that are likely valid and coherent with the training data.

2.1 Related Work

Structured generative tasks have traditionally been tackled using probabilistic graphical models [58] and grammars [116], which lack support for representation learning and efficient sampling under constraints. Tractable probabilistic circuits [88, 57] are a recent alternative that make use of ideas from knowledge compilation [17] to provide efficient generation of valid structures. These approaches generate valid objects by constructing a circuit (a polynomial) that encodes both the hard constraints and the probabilistic structure of the

[1]With high probability. Invalid structures, when generated, can be checked and rejected efficiently. In this sense, CANs are related to learning efficient proposal distributions [3].

problem. Although inference is linear in the size of the circuit, the latter can grow very large if the constaints are complex enough. In contrast, CANs model the probabilistic structure of the problem using a neural architecture, while relying on knowledge compilation for encoding the hard constraints during training. Moreover, the circuit can be discarded at inference time. The time and space complexity of sampling for CANs is therefore roughly independent from the complexity of the constraints in practice.

Deep generative models developed for structured tasks are special-purpose, in that they rely on ad-hoc architectures, tackle specific applications, or have no support for efficient sampling [32, 18, 140, 117]. Some recent approaches have focused on incorporating a constraint learning component in training deep generative models, using reinforcement learning [18] or inverse reinforcement learning [42] techniques. This direction is complementary to ours and is useful when constraints are not known in advance or cannot be easily formalized as functions of the generator output. Indeed, our experiment on molecule generation shows the advantages of enriching CANs with constraint learning to generate high quality and diverse molecules.

Other general approaches for injecting knowledge into neural nets (like deep statistical-relational models [69, 73, 76], tensor-based models [94, 19], and fuzzy logic-based models [75]) are either not generative or require the constraints to be available at inference time.

2.2 Unconstrained GANs

GANs [31] are composed of two neural nets: a discriminator d trained to recognize "real" objects $\mathbf{x} \in \mathcal{X}$ sampled from the data distribution P_r, and a generator $g : \mathcal{Z} \to \mathcal{X}$ that maps random latent vectors $\mathbf{z} \in \mathcal{Z}$ to objects $g(\mathbf{x})$ that fool the discriminator. Learning equates to solving the minimax game

$\min_g \max_d \; f_{\text{GAN}}(g, d)$ with value function:

$$f_{\text{GAN}}(g, d) := \mathbb{E}_{\mathbf{x} \sim P_r}[\log P_d(\mathbf{x})] + \mathbb{E}_{\mathbf{x} \sim P_g}[\log(1 - P_d(\mathbf{x}))] \tag{2.1}$$

Here $P_g(\mathbf{x})$ and $P_d(\mathbf{x}) := P_d(\text{real} \,|\, \mathbf{x})$ are the distributions induced by the generator and discriminator, respectively. New objects $\mathbf{x}$ can be sampled by mapping random vectors $\mathbf{z}$ using the generator, i.e., $\mathbf{x} = g(\mathbf{z})$. Under idealized assumptions, the learned generator matches the data distribution:

Theorem 1 ([31]). *If g and d are non-parametric and the leftmost expectation in Eq. 2.1 is approximated arbitrarily well by the data, the global equilibrium (g^*, d^*) of Eq. 2.1 satisfies $P_{d^*} \equiv \frac{1}{2}$ and $P_{g^*} \equiv P_r$.*

In practice, training GANs is notoriously hard [98, 78]. The most common failure mode is mode collapse, in which the generated objects are clustered in a tiny region of the object space. Remedies include using alternative objective functions [31], divergences [82, 2] and regularizers [79]. In our experiments, we apply some of these techniques to stabilize training.

In structured tasks, the objects of interest are usually discrete. In the following, we focus on stochastic generators that output a *categorical distribution* $\boldsymbol{\theta}(\mathbf{z})$ over $\mathcal{X}$ and objects are sampled from the latter. In this case, $P_g(\mathbf{x}) = \int_{\mathcal{Z}} P_g(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) d\mathbf{z} = \int_{\mathcal{Z}} \boldsymbol{\theta}(\mathbf{z}) p(\mathbf{z}) d\mathbf{z} = \mathbb{E}_{\mathbf{z}}[\boldsymbol{\theta}(\mathbf{z})]$.

2.3 Generating Structures with CANs

Our goal is to learn a deep generative model that outputs structures $\mathbf{x}$ consistent with validity constraints and an unobserved distribution P_r. We assume to be given: i) a feature map $\phi : \mathcal{X} \to \{0, 1\}^b$ that extracts b binary features from $\mathbf{x}$, and ii) a single validity constraint ψ encoded as a Boolean formula on $\phi(\mathbf{x})$. If $\mathbf{x}$ is binary, ϕ can be taken to be the identity; later we will discuss some alternatives. Any discrete structured space can be encoded this way.

2.3.1 Limitations of GANs

Standard GANs struggle to output valid structures, for two main reasons. First, the number of examples necessary to capture any non-trivial constraint ψ can be intractably large.[2] This rules out learning the rules of chemical validity or, worse still, graph reachability from even moderately large data sets. Second, in many cases of interest the examples are noisy and do violate ψ, in which case the data lures GANs into learning *not* to satisfy the constraint:

Corollary 1. *Under the assumptions of Theorem 1, given a target distribution P_r, a constraint ψ consistent with it, and a dataset of examples $\mathbf{x}$ sampled i.i.d. from a corrupted distribution $P_r{}' \neq P_r$ inconsistent with ψ, GANs associate non-zero mass to infeasible objects.*

This follows easily from Theorem 1, as the optimal generator satisfies $P_g \equiv P_r{}'$, which is inconsistent with ψ. Since Theorem 1 captures the *intent* of GAN training, this corollary shows that GANs are *by design* incapable of handling invalid examples.

2.3.2 Constrained Adversarial Networks

Constrained Adversarial Networks (CANs) avoid these issues by taking both the data and the target structural constraint ψ as inputs. The value function is designed so that the generator maximizes the probability of generating valid structures. In order to derive CANs it is convenient to start from the following alternative GAN value function [31]: $f_{\text{ALT}}(g, d) := \mathbb{E}_{\mathbf{x}\sim P_r}[\log P_d(\mathbf{x})] - \mathbb{E}_{\mathbf{x}\sim P_g}[\log P_d(\mathbf{x})]$.

Let (g, d) be a GAN and $v(\mathbf{x}) = \mathbb{1}\{\phi(\mathbf{x}) \models \psi\}$ be a fixed discriminator that distinguishes between valid and invalid structures, where $\models$ indicates

[2]The VC dimension of unrestricted discrete formulas is exponential in the number of variables [124].

logical entailment. Ideally, we wish the generator to *never* output invalid structures. This can be achieved by using an aggregate discriminator $a(\mathbf{x})$ that only accepts configurations that are both valid and high-quality w.r.t. d. Let A be the indicator that a classifies $\mathbf{x}$ as real, and similarly for D and V. By definition:

$$P_a(\mathbf{x}) = P(A \,|\, \mathbf{x}) = P(D \,|\, V, \mathbf{x}) P(V \,|\, \mathbf{x}) = P_d(\mathbf{x}) \mathbb{1}\{\phi(\mathbf{x}) \models \psi\} \tag{2.2}$$

Plugging the aggregate discriminator into the alternative value function gives:

$$\begin{aligned}
&\arg\max_a \; f_{\text{ALT}}(g, a) && (2.3)\\
&= \arg\max_d \; \mathbb{E}_{P_r}[\log P_d(\mathbf{x}) + \log \mathbb{1}\{\phi(\mathbf{x}) \models \psi\}] - \mathbb{E}_{P_g}[\log P_d(\mathbf{x}) + \log \mathbb{1}\{\phi(\mathbf{x}) \models \psi\}] && (2.4)\\
&= \arg\max_d \; \mathbb{E}_{P_r}[\log P_d(\mathbf{x})] - \mathbb{E}_{P_g}[\log P_d(\mathbf{x})] - \mathbb{E}_{P_g}[\log \mathbb{1}\{\phi(\mathbf{x}) \models \psi\}] && (2.5)\\
&= \arg\max_d \; f_{\text{ALT}}(g, d) - \mathbb{E}_{P_g}[\log \mathbb{1}\{\phi(\mathbf{x}) \models \psi\}] && (2.6)
\end{aligned}$$

The second step holds because $\mathbb{E}_{P_r}[\log \mathbb{1}\{\phi(\mathbf{x}) \models \psi\}]$ does not depend on d. If g allocates non-zero mass to *any* measurable subset of invalid structures, the second term becomes $+\infty$. This is consistent with our goal but problematic for learning. A better alternative is to optimize the lower bound:

$$SL_\psi(g) := -\log P_g(\psi) = -\log \mathbb{E}_{P_g}[\mathbb{1}\{\phi(\mathbf{x}) \models \psi\}] \leq -\mathbb{E}_{P_g}[\log \mathbb{1}\{\phi(\mathbf{x}) \models \psi\}] \tag{2.7}$$

This term is the *semantic loss* (SL) proposed in [139] to inject knowledge into neural networks. The SL is much smoother than the original and it only evaluates to $+\infty$ if P_g allocates *all* the mass to infeasible configurations. This immediately leads to the CAN value function:

$$f_{\text{CAN}}(g, d) := f_{\text{ALT}}(g, d) + \lambda SL_\psi(g) \tag{2.8}$$

where $\lambda > 0$ is a hyper-parameter controlling the importance of the constraint. This formulation is related to integral probability metric-based GANs, cf. [63]. The SL can be viewed as the negative log-likelihood of ψ, and hence it rewards the generator proportionally to the mass it allocates to valid structures. The expectation in Eq. 2.7 can be rewritten as:

$$\mathbb{E}_{\mathbf{x}\sim P_g}[\mathbb{1}\{\phi(\mathbf{x}) \models \psi\}] = \sum_{\mathbf{x}:\phi(\mathbf{x})\models\psi} P_g(\mathbf{x}) = \mathbb{E}_{\mathbf{z}}\left[\sum_{\mathbf{x}:\phi(\mathbf{x})\models\psi} \prod_{i\,:\,x_i=1} \theta_i(\mathbf{z}) \prod_{i\,:\,x_i=0} (1-\theta_i(\mathbf{z}))\right] \tag{2.9}$$

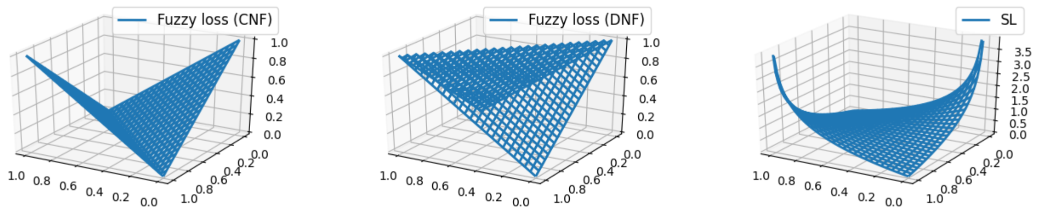

Figure 2.1: Left: fuzzy logic encoding (using the Łukasiewicz T-norm) of $x \oplus y$ in CNF format as a function of $P(x = 1)$ and $P(y = 1)$. Middle: encoding of DNF XOR. Right: SL of either encoding.

Hence, the SL is the negative logarithm of a polynomial in $\boldsymbol{\theta}$ and it is fully differentiable.[3] In practice, below we apply the semantic loss term directly to f_{GAN}, i.e., $f_{\text{CAN}}(g, d) := f_{\text{GAN}}(g, d) + \lambda SL_\psi(g)$.

If the SL is given large enough weight λ then it gets closer to the ideal "hard" discriminator, and therefore more strongly encourages the CAN to generate valid structures. Under the preconditions of Theorem 1, it is clear that for $\lambda \to \infty$ CANs generate valid structures only:

Proposition 1. *Under the assumptions of Corollary 1, CANs associate zero mass to infeasible objects, irrespective of the discrepancy between P_r and $P_r{}'$.*

Indeed, any global equilibrium (g^*, d^*) of $\min_g \max_d f_{\text{CAN}}(g, d)$ minimizes the second term: the minimum is attained by $\log P_{g^*}(\psi) = 0$, which entails $P_{g^*}(\neg\psi) = 0$. Of course, as with standard GANs, the prerequisites are often violated in practice. Regardless, Proposition 1 works as a sanity check, and shows that, in contrast to GANs, CANs are appropriate for structured generative tasks.

A possible alternative to the SL is to introduce a differentiable knowledge-based loss into the value function by relaxing the constraint ψ using fuzzy logic, as done in a number of recent works on discriminative deep learning [19, 75]. Apart from lacking a formal derivation in terms of expected probability of

[3]As long as $P_g(\mathbf{x}) > 0$, which is always the case in practice.

satisfying constraints, the issue is that fuzzy logic is not semantically sound, meaning that equivalent encodings of the same constraint may give different loss functions [29]. Figure 2.1 illustrates this on an XOR constraint: the "fuzzy loss" and its gradient change radically depending on whether the XOR is encoded as CNF (left) or DNF (middle), while the SL is unaffected (right).

Evaluating the Semantic Loss The sum in Eq. 2.9 is the unnormalized probability of generating a valid configuration. Evaluating it requires to sum over *all* solutions of ψ, weighted according to their probability with respect to $\boldsymbol{\theta}$. This task is denoted Weighted Model Counting (WMC) [7]. Naïvely implementing WMC is infeasible in most cases, as it involves summing over exponentially many configurations. Knowledge compilation (KC) [17] is a well known approach in automated reasoning and solving WMC through KC is the state-of-the-art for answering probabilistic queries in discrete graphical models [7, 22, 123]. Roughly speaking, KC leverages distributivity to rewrite the polynomial in Eq. 2.9 as compactly as possible, often offering a tremendous speed-up during evaluation. This is achieved by identifying shared sub-components and compactly representing the factorized polynomial using a DAG. Target representations for the DAG (OBDDs, DNNFs, *etc.* [17]) differ in succinctness and enable different polytime (in the size of the DAG) operations. As done in [139], we compile the SL polynomial into a Sentential Decision Diagram (SDD) [16] that enables efficient WMC and therefore exact evaluation of the SL and of its gradient. KC is key in making evaluation of the Semantic Loss and of its gradient practical, at the cost of an offline compilation step – which is however performed only once before training.

The main downside of KC is that, depending on the complexity of ψ, the compiled circuit may be large. This is less of an issue during training, which is often performed on powerful machines, but it can be problematic for inference, especially on embedded devices. A major advantage of CANs is that the circuit is not required for inference (as the latter consists of a simple forward pass over the generator), and can thus be thrown away after training. This means that CANs incur no space penalty during inference compared to GANs.

The embedding function ϕ The embedding function $\phi(\mathbf{x})$ extracts Boolean variables to which the SL is then applied. In many cases, as in our molecule experiment, ϕ is simply the identity map. However, when fed a particularly complex constraint ψ, KC may output an SDD too large even for the training stage. In this case, we use ϕ to map $\mathbf{x}$ to an application-specific embedding space where ψ (and hence the SL polynomial) is expressible in compact form. We successfully employed this technique to synthesize Mario levels where the goal tile is reachable from the starting tile; all details are provided below. The same technique can be exploited for dealing with other complex logical formulas beyond the reach of state-of-the-art knowledge compilation.

2.3.3 Conditional CANs

So far we described how to use the SL for enforcing structural constraints on the generator's output. Since the SL can be applied to any distribution over binary variables, it can also be used to enforce conditional constraints that can be turned on and off at inference time. Specifically, we notice that the constraint can involve also latent variables, and we show how this can be leveraged for different purposes. Similarly to InfoGANs [10], the generator's input is augmented with an additional binary vector $\mathbf{c}$. Instead of maximizing (an approximation of) the mutual information between $\mathbf{c}$ and the generator's output, the SL is used to logically bind the input codes to semantic features or constraint of interest. Let $\psi_1, \ldots, \psi_k$ be k constraints of interest. In order to make them switchable, we extend the latent vector $\mathbf{z}$ with k fresh variables

$\mathbf{c} = (c_1, \ldots, c_k) \in \{0,1\}^k$ and train the CAN using the constraint:

$$\psi = \bigwedge_{i=1}^{k} (c_i \leftrightarrow \psi_i)$$

where the prior $P(\mathbf{c})$ used during training is estimated from data.

Using a conditional SL term during training results in a model that can be conditioned to generate object with desired, arbitrarily complex properties ψ_i at inference time. Additionally, this feature shows a beneficial effect in mitigating mode collapse during training, as reported in Section 2.4.2.

2.4 Experiments

Our experimental evaluation aims at answering the following questions:

Q1 Can CANs with tractable constraints achieve better results than GANs?

Q2 Can CANs with intractable constraints achieve better results than GANs?

Q3 Can constraints be combined with rewards to achieve better results than using rewards only?

We implemented CANs[4] using Tensorflow and used PySDD[5] to perform knowledge compilation. We tested CANs using different generator architectures on three real-world structured generative tasks.[6] In all cases, we evaluated the objects generated by CANs and those of the baselines using three metrics (adopted from [100]): **validity** is the proportion of sampled objects that are valid; **novelty** is the proportion of valid sampled objects that are not present in the training data; and **uniqueness** is the proportion of valid unique (non-repeated) sampled objects.

[4]The code is freely available at https://github.com/unitn-sml/CAN

[5]URL: `pypi.org/project/PySDD/`

[6]Details can be found in Appendix A.

2.4.1 Super Mario Bros level generation

In this experiment we show how CANs can help in the challenging task of learning to generate videogame levels from user-authored content. While procedural approaches to videogame level generation have successfully been used for decades, the application of machine learning techniques in the creation of (functional) content is a relatively new area of research [112]. On the one hand, modern video game levels are characterized by aesthetical features that cannot be formally encoded and thus are difficult to implement in a procedure, which motivates the use of ML techniques for the task. On the other hand, the levels have often to satisfy a set of functional (hard) constraints that are easy to guarantee when the generator is hand-coded but pose challenges for current machine learning models.

Architectures for Super Mario Bros level generation include LSTMs [111], probabilistic graphical models [33], and multi-dimensional MCMC [109]. MarioGANs [117] are specifically designed for level generation, but they only constrain the mixture of tiles appearing in the level. This technique cannot be easily generalized to arbitrary constraints.

In the following, we show how the semantic loss can be used to encode useful hard constraints in the context of videogame level generation. These constraints might be functional requirements that apply to every generated object or might be contextually used to steer the generation towards objects with certain properties. In our empirical analysis, we focus on *Super Mario Bros* (SMB), possibly one of the most studied video games in tile-based level generation.

Recently, [127] applied Wasserstein GANs (WGANs) [2] to SMB level generation. The approach works by first training a generator in the usual way, then using an evolutionary algorithm called Covariance Matrix Adaptation Evolution Strategy (CMA-ES) to search for the best latent vectors according to a user-defined fitness function on the corresponding levels. We stress that this technique is orthogonal to CANs and the two can be combined together. We adopt the same experimental setting, WGAN architecture and training procedure of [127]. The structured objects are 14×28 tile-based representations of SMB levels (e.g. Fig. 2.2) and the training data is obtained by sliding a 28 tiles window over levels from the *Video game level corpus* [113].

We run all the experiments on a machine with a single 1080Ti GPU for 4 times with random seeds.

CANs with tractable constraints: generating SMB levels with *pipes*

In this experiment, the focus is on showing how CANs can effectively deal with constraints that can be directly encoded over the generator output. Pipes are made of four different types of tiles. They can have a variable height but the general structure is always the same: two tiles (*top-left* and *top-right*) on top and one or more pairs of body tiles (*body-left* and *body-right*) below (see the `CAN - pipes` in picture in Fig. 2.2 for examples of valid pipes). Since encoding all possible dispositions and combinations of pipes in a level would result in an extremely large propositional formula, we apply the constraint locally to a 2×2 window that is slid, horizontally and vertically, by one tile at a time (notice that all structural properties of pipes are covered using this method). The constraint consists of a lot of implications of the type "if this is a *top-left* tile, then the tile below must be a *body-left* one" conjoined together (see the Appendix A for the full formula). The relative importance of the constraints is determined by the hyper-parameter λ (see Eq. 2.8).

There are two major problems in the application of the constraint on pipes when using a large λ: i) *vanishing pipes*: this occurs because the generator can satisfy the constraint by simply generating layers without pipes; ii) *mode collapse*: the generator may learn to place pipes always in the same positions.

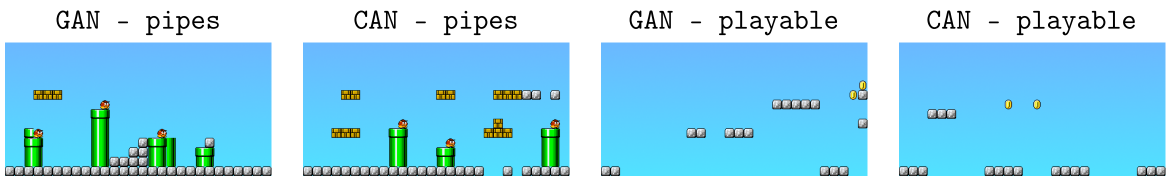

Figure 2.2: Examples of SMB levels generated by GAN and CAN. Left: generating levels containing pipes; right: generating reachable levels. For each of the two settings we report prototypical examples of levels generated by GAN (first and third picture) and CAN (second and fourth picture). Notice how all pipes generated by CAN are valid, contrarily to what happens for GAN, and that the GAN generates a level that is not playable (because of the big jump at the start of the map).

We address both issues by introducing the SL after an initial bootstrap phase (of $5,000$ epochs) in which the generator learns to generate sensible objects, and by linearly increasing its weight from zero to $\lambda = 0.2$. The final value for λ was chosen as the highest value allowing to retain al least 80% of pipe tiles on average with respect to a plain GAN. All experiments were run for $12,000$ epochs.

Table 2.1 reports experimental results comparing GAN and CAN trained on all levels containing pipes. CAN manage to almost double the validity of the generated levels (see the two left pictures in Fig. 2.2 for some prototypical examples) while retaining about 82% of the pipe tiles and without any significant loss in terms of diversity (as measured by the L1 norm on the difference between each pair of levels in the generated batch) or cost in terms of training (roughly doubled training times). Inference is real-time (< 40 ms) for both architectures.

These results allow to answer **Q1** affirmatively.

Model	# Maps	Validity (%)	Average pipe-tiles / level	L1 Norm	Training time
GAN	7	47.6 ± 8.3	7.8	0.0115	1h 12m
CAN	7	83.2 ± 4.8	6.4	0.0110	2h 2m

Table 2.1: Comparison between GAN and CAN on SMB level generation with pipes. The 7 maps containing pipes are *mario-1-1*, *mario-2-1*, *mario-3-1*, *mario-4-1*, *mario-4-2*, *mario-6-2* and *mario-8-1*, for a total of 1,404 training samples. Results report validity, average number of pipe tiles per level, L1 norm on the difference between each pair of levels in the generated batch and training time. Inference is real-time (< 40 ms) for both architectures.

CANs with intractable constraints: generating *playable* SMB levels

In the following we show how CANs can be successfully applied in settings where constraints are too complex to be directly encoded onto the generator output. A level is *playable* if there is a feasible path[7] from the left-most to the right-most column of the level. We refer to this property as *reachability*. We compare CANs with CMA-ES, as both techniques can be used to steer the network towards the generation of playable levels. In CMA-ES, the fitness function doesn't have to be differentiable and the playability is computed on the output of an A* agent (the same used in [127]) playing the level. Having the SL to steer the generation towards playable levels is not trivial, since it requires a differentiable definition of playability. Directly encoding the constraint in propositional logic is intractable. Consider the size of a first order logic propositional formula describing all possible path a player can follow in the level. We thus define the playability constraint on the output of an embedding function ϕ (modelled as a feedforward NN) that approximates tile reachability. The function is trained to predict whether each tile is reachable from the left-most column using traces obtained from the A* agent. See Appendix A for the details.

Table 2.2 shows the validity of a batch of 1,000 levels generated respectively

[7]According to the game's physics.

Network type	Level	Tested samples	Validity	Training time	Inference time per sample
GAN	mario-1-3	1000	9.80%	1 h 15 min	$\sim$ 40 ms
GAN + CMA-ES	mario-1-3	1000	65.90%	1 h 15 min	$\sim$ 22 min
CAN	mario-1-3	1000	71.60%	1 h 34 min	$\sim$ 40 ms
GAN	mario-3-3	1000	13.00%	1 h 11 min	$\sim$ 40 ms
GAN + CMA-ES	mario-3-3	1000	64.20%	1 h 11 min	$\sim$ 22 min
CAN	mario-3-3	1000	62.30%	1 h 27 min	$\sim$ 40 ms

Table 2.2: Results on the generation of *playable* SMB level. Levels *mario-1-3* (123 training samples) and *mario-3-3* (122 training samples) were chosen due to their high solving complexity. Results compare a baseline GAN, a GAN combined with CMA-ES and a CAN. Validity is defined as the ability of the A* agent to complete the level. Note that inference time for GAN and CAN is measured in milliseconds while time for GAN + CMA-ES is in minutes.

by plain GAN, GAN combined with CMA-ES using the default parameters for the search, and a forward pass of CAN. Each training run lasted 15000 epochs with all the default hyper parameters defined in [127], and the SL was activated from epoch 5000 with $\lambda = 0.01$, which validation experiments showed to be a reasonable trade-off between SL and generator loss. Results show that CANs achieves better (*mario-1-3*) or comparable (*mario-3-3*) validity with respect to GAN + CMA-ES at a fraction of the inference time. At the cost of pretraining the reachability function, CANs avoid the execution of the A* agent during the generation and sample high quality objects in milliseconds (as compared to minutes), thus enabling applications to create new levels at run time. Moreover, no significant quality degradation can be seen on the generated levels as compared to the ones generated by plain GAN (which on the other hand fails most of the time to generate reachable levels), as can be seen in Fig. 2.2. With these results, we can answer **Q2** affirmatively.

2.4.2 Molecule generation

Most approaches in molecule generation use variational autoencoders (VAEs) [30, 60, 15, 99], or more expensive techniques like MCMC [101]. Closest to CANs are ORGANs [32] and MolGANs [18], which respectively combine Sequence GANs (SeqGANs) and Graph Convolutional Networks (GCNs) with a reward network that optimizes specific chemical properties. Albeit comparing favorably with both sequence models [49, 32] (using SMILE representations) and likelihood-based methods, MolGAN are reported to be susceptible to mode collapse.

In this experiment, we investigate **Q3** by combining MolGAN's adversarial training and *reinforcement learning objective* with a conditional SL term on the task of generating molecules with certain desirable chemical properties. In contrast with our previous experimental settings, here the structured objects are undirected graphs of bounded maximum size, represented by discrete tensors that encode the atom/node type (padding atom (no atom), Carbon, Nitogren, Oxygen, Fluorine) and the bound/edge type (padding bond (no bond), single, double, triple and aromatic bond). During training, the network implicitly rewards validity and the maximization of the three chemical properties at once: **QED** (druglikeness), **SA** (synthesizability) and **logP** (solubility). The training is stopped once the uniqueness drops under 0.2. We augment the MolGAN architecture with a conditional SL term, making use of 4 latent dimensions to control the presence of one of the 4 types of atoms considered in the experiment, as shown in Section 2.3.3.

Conditioning the generation of molecules with specific atoms at training time mitigates the drop in uniqueness caused by the reward network during the training. This allows the model to be trained for more epochs and results in more diverse and higher quality molecules, as reported in Table 2.3.

In this experiment, we train the model on a NVIDIA RTX 2080 Ti. The total training time is around 1 hour, and the inference is real-time. Using CANs produced a negligible overhead during the training with respect to the original model, providing further evidence that the technique doesn't heavily impact on the training. This results suggest that coupling CANs with

Reward for	SL	validity	uniqueness	diversity	QED	SA	logP
QED + SA + logP	False	97.4	2.4	91.0	47.0	84.0	65.0
	True	96.6	2.5	98.8	51.8	90.7	73.6

Table 2.3: Results of using the semantic loss on the MolGAN architecture. The diversity score is obtained by comparing sub-structures of generated samples against a random subset of the dataset. A lower score indicates a higher amount of repetitions between the generated samples and the dataset. The first row refers to the results reported in the MolGAN paper.

a reinforcement learning objective is beneficial, answering **Q3** affirmatively.

2.5 Conclusion

We presented Constrained Adversarial Networks (CANs), a generalization of GANs in which the generator is encouraged *during training* to output valid structures. CANs make use of the semantic loss [139] to penalize the generator proportionally to the mass it allocate to invalid structures and. As in GANs, generating valid structures (on average) requires a simple forward pass on the generator. Importantly, the data structures used by the SL, which can be large if the structural constraints are very complex, are discarded after training. CANs were proven to be effective in improving the quality of the generated structures without significantly affecting inference run-time, and conditional CANs proved useful in promoting diversity of the generator's outputs.

Chapter 3

Semantic-Guided Image Inpainting

Manipulating images to insert and remove objects automatically is of paramount relevance for a large number of real-world applications including data augmentation, photo editing and Augmented Reality (AR). Recent literature has shown promising results in image manipulation to translate images from one domain to another [47, 152, 12, 71, 106, 107, 70], inpaint missing parts of images [144, 80, 50], and change portion of faces and fashion garments [34, 50]. However, manipulating images (i.e. adding, reconstructing and removing objects) of complex scenes is a challenging and still unsolved problem. Complex scenes are indeed characterized by multiple semantics and objects, which are often cluttered or occluded, making it difficult to apply conventional image inpainting techniques.

Existing works on image editing focus either on object insertion or removal. Solutions to the former task usually require users to draw segmentation pixels [40] or a precise bounding box of the object to be inserted [62], while object removal has been addressed by image inpainting [144, 80, 50], which reconstructs the most probable pattern from contextual pixels. Thus, one has to choose between models that insert objects in controlled settings and models that inpaint corrupted areas. Moreover, complex scenes (e.g. urban scenes) are often overlooked in favour of natural scenes and photographs

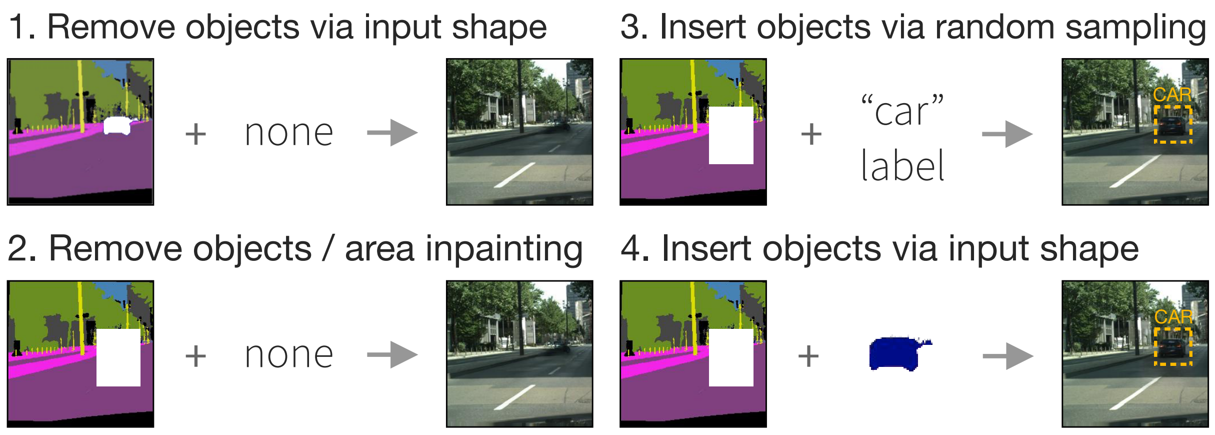
1. Remove objects via input shape
+ none
3. Insert objects via random sampling
+ "car" label
CAR
2. Remove objects / area inpainting
+ none
4. Insert objects via input shape
+
CAR

Figure 3.1: Our holistic model can be applied in a wide range of manipulations in complex scenes. At inference time users can remove objects by either precisely indicating a mask or an entire area. Moreover, they can insert new objects (e.g. cars, pedestrians) by randomly generating them or by feeding a segmentation mask as input.

with a small number of semantic classes, which allows the use of different stratagems to guide the generative network. In non-complex scenes, literature relies on edge maps [80] and object contours [138], which results are however not satisfactory when the missing region is large or complex.

In this chapter, we focus on complex urban scenes that contain multiple objects, clutter and numerous semantic classes. We propose a novel and unified framework to manipulate images by removing and inserting objects. We formulate the problem by learning to reconstruct (inpaint) missing regions and generate plausible shapes of objects to be inserted. To help the model at understanding complex scenes, we leverage information from semantic segmentation maps in order to guide the network in both the encoding and the decoding phases. We learn to accurately generate both the semantic segmentation and the real pixels enforcing them to be consistent with each other. To this end, we design and propose a novel decoder module based on Spatially-Adaptive (DE) normalization (SPADE) [86] that uses the predicted segmentation to normalize generated features and synthesize high-quality images. Differently from previous works, our unified framework allows numerous use cases at inference time. For example, users can interact with the model by precisely removing the pixel of an object, removing entire areas, inserting random sampled objects or placing input shapes directly in the scene (see Figure 3.1).

To evaluate our framework, we conduct experiments on two large-scale datasets of urban scenes, namely Cityscapes [14] and Indian Driving [125]. These datasets consist of a large number of semantic classes (on average 17) and diverse and unstructured environmental conditions, which make them a perfect benchmark for complex scene manipulations. Our results show that with the proposed method we can insert and place objects in existing complex scenes generating high-quality images, outperforming state-of-the-art approaches on image inpainting.

Overall, the main contributions of our work are as follows:

- We propose a new holistic framework to manipulate complex scenes and allow users to insert and remove different types of objects in a single pass. At inference time, users can do a wide range of manipulations, considering different types of object insertion and removal operations.

- We design a new decoder module based on SPADE [86] that uses the predicted segmentation map instead of the ground truth. Thus, at inference time we allow the use of SPADE without asking the user to specify the precise semantic pixels of the desired transformation.

- We validate the proposed solution in the challenging task of manipulation of urban scenes using two large-scale datasets, namely Cityscapes and Indian Driving. Quantitative and qualitative results show that our method significantly outperforms the state of the art models in all the experiments.

3.1 Related Work

Our work is best placed in the literature of image inpainting and image manipulation. The former aims at restoring a damaged image or remove undesired objects, while the latter tries to synthetize new images with a user-specified object.

Image inpainting. Image inpainting approaches have witnessed a dramatic improvement in image quality especially thanks to deep learning methods, in particular to Generative Adversarial Networks (GANs) [31]. Most notably, Pathak *et al.* [87] propose an encoder-decoder network, inspired by auto-encoder approaches, which synthetizes a part region of the image depending on its surroundings. The combination of reconstruction and adversarial losses are shown to be sufficient for the neural networks to inpaint the missing region of the image. However, blurry results and disconnected edges frequently occur in generated regions. Thus, Iizuka *et al.* [43] propose to use dilated convolutions to increase the receptive field and combine the global and local discriminators to improve the quality of the generated patches. Yu *et al.* [143] instead propose the use of a two-stage approach to first coarsely reconstruct the image and then refine the coarse details. An additional contextual attention module is proposed to capture distant information. Zheng *et al.* [150] focus on generating multiple plausible results for the same missing region by using a Variational-Autoencoder (VAE). Finally, Yu *et al.* [144] propose a spatial region-wise approach that normalizes the corrupted and uncorrupted regions with two different means and variances.

Recently, various efforts using structural information have been explored to better reconstruct edges and contours. For example, Nazeri *et al.* [80] and Jo *et al.* [50] use input edge maps during the image inpainting. Xiong *et al.* [138] use object contours and a multi-stage process to disentangle the background from a foreground object. Ren *et al.* [93] instead focus on reconstructing missing structures of free-form missing parts through a flow-based method. Song *et al.* [110] explore semantic segmentation images to guide the reconstruction in a two-stage process. However, most of the techniques found to be effective in non-complex scenes (e.g. faces, simple scenes) result in non-satisfactory reconstructions due to the higher clutter and number of different semantics of complex scenes. Moreover, existing approaches based on image inpainting remove existing objects and reconstruct the background patterns without inserting new objects.

Image manipulation. Image manipulation aims at modifying an existing image towards a user-desired outcome. In image-to-image translation this outcome is usually changing the visual appearance of an image while maintaining the original structure (e.g. blonde hair woman $\leftrightarrow$ black hair man). For example, literature has shown promising results on translating domains in the image space (e.g. blonde hair woman to black hair woman) [47, 152, 12, 71], and from the image space to the semantic space [47, 131]. There has been also some efforts towards free-form editing, where the mask is not rectangular nor regular. Notable example is free-form editing of faces and fashion pictures where people draw a sketch of the desired transformation [21, 50].

However, synthesising new object instances in an existing complex scene is a challenging task as it involves the generation of a reliable structure and appearance, which has to fit harmoniously with the structure of the rest of the image. This problem has been much less studied in the literature. Ouyang *et al.* [84] propose a conditional GAN operating on the image manifold to insert plausible pedestrians into an urban scene. Hong *et al.* [40] instead suggest to focus on the semantic segmentation, inpaint missing regions and paste into them a segmentation mask provided by the user as input. Lee *et al.* [62] propose a network that predicts the location and the shape of different objects (namely pedestrian and car) to insert them in the semantic segmentation

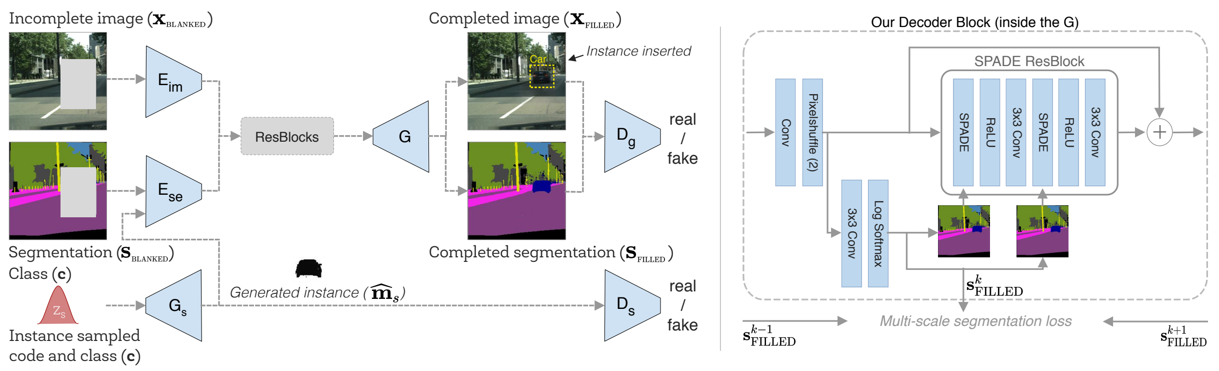
Incomplete image ($\mathbf{x}_{\text{BLANKED}}$)
E_{im}
E_{se}
ResBlocks
G
Completed image ($\mathbf{x}_{\text{FILLED}}$)
Instance inserted
Car
D_g
real / fake
Segmentation ($\mathbf{s}_{\text{BLANKED}}$)
Class (c)
z_s
Instance sampled code and class (c)
G_s
Generated instance ($\widehat{\mathbf{m}}_s$)
Completed segmentation ($\mathbf{s}_{\text{FILLED}}$)
D_s
real / fake
Our Decoder Block (inside the G)
Conv
Pixelshuffle (2)
SPADE ResBlock
SPADE
ReLU
3x3 Conv
SPADE
ReLU
3x3 Conv
3x3 Conv
Log Softmax
$\mathbf{s}^k_{\text{FILLED}}$
$\mathbf{s}^{k-1}_{\text{FILLED}}$
Multi-scale segmentation loss
$\mathbf{s}^{k+1}_{\text{FILLED}}$

Figure 3.2: (left) Our model synthetizes a new image ($\boldsymbol{x}_{\text{FILLED}}$) and its segmentation ($\boldsymbol{s}_{\text{FILLED}}$) from an incomplete image ($\boldsymbol{x}_{\text{BLANKED}}$), its segmentation map ($\boldsymbol{s}_{\text{BLANKED}}$) and an optional instance, which might be sampled from a latent distribution ($\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$) or fed in input. The two discriminators encourage the generation of samples that resemble the real data distribution. (right) Our decoder block in the generator is based on SPADE [86]. We propose a modification to use the predicted segmentation map instead of the ground truth segmentation and adopt a multi-scale segmentation loss to better learn it.

space. Finally, Berlincioni *et al.* [4] focus on semantic segmentation removing cars and pedestrians from a road layout by feeding the binary masks in input to the model. However, existing work allow users to either insert or delete objects from an existing scene. Interestingly, most of the literature focus just on the semantic segmentation space leaving to another network (e.g. pix2pix [47]) the generation of real pixels. This results in a fragmented approach that makes it difficult to edit an existing image. To the best of our knowledge, our approach is the first which permits to model multiple object instances' types and which allows, at the same time, to remove and reconstruct portions of images.

From layout to images. Noteworthy are the preliminary results in the related task of layout-to-image generation, which starts from a sparse object layout to synthetize plausible images. Pavllo *et al.* [114] created a layout-to-mask-to-image system starting from a bounding box layout to generate plausible segmentation maps and images. Zhao *et al.* [149] started from a scene description graph to generate possible images that correspond to the input graph. These works focus on synthetizing plausible images from an empty frame either using segmentation masks or bounding box layouts. However, in our setting we want to generate new objects and reconstruct images that have to fit and adapt well in the existing complex scene. Moreover, most of the layout-to-mask-to-image works test only few semantics and objects, while in our setting there are on average 17 different object categories per image.

3.2 Approach

In this chapter, we aim at editing an image by removing and inserting object instances (e.g. car or pedestrian) in an existing urban context. At inference time, users can thus remove existing objects by blanking-out a portion of the image and asking the network to insert an object instance to replace the removed one (see Figure 3.1).

Inspired by recent literature of image inpainting, we propose a one-stage deep architecture that predicts the missing parts of the given images but also inserts new object instances. Our model allows the user to specify the exact position of the object or to let the network decide where to insert it inside a bigger area. We learn a latent space of object shapes from which sample the plausible objects to be inserted. To help the network dealing with this challenging task, we guide the generation of new images through semantic segmentation. Figure 3.2 shows an overview of the proposed network.

Formally, given an image $\boldsymbol{x}_{\text{BLANKED}} \in \mathbb{R}^{H \times W \times 3}$, its segmentation map $\boldsymbol{s}_{\text{GT}} \in \mathbb{R}^{H \times W \times C}$ (C is the number of classes in the segmentation map), a latent code $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$ and a one-hot encoding label $\boldsymbol{c} \in \mathbb{R}^{D}$, we want to synthetize a new image $\boldsymbol{x}_{\text{FILLED}}$. Instead of the missing area, this image has to contain an object instance of class $\boldsymbol{c}$ and to have the missing part reconstructed resembling the ground truth image $\boldsymbol{x}_{\text{GT}} \in \mathbb{R}^{H\times W\times 3}$, which is the original image without the blanked-out area.

Training the network. For a pair of images ($\boldsymbol{x}_{\text{GT}}$, $\boldsymbol{s}_{\text{GT}}$) in the dataset, we randomly generate a binary image mask $\boldsymbol{m} \in \{0,1\}^{H\times W\times 1}$ that defines the area of the original image to be blanked out. Whenever the mask $\boldsymbol{m}$ contains an object belonging to the modelled classes (e.g. car and pedestrian) we extract also the binary mask $\boldsymbol{m}_s \in \{0,1\}^{H\times W\times 1}$ that is 0 everywhere except in the pixel locations of the object. We note that the mask $\boldsymbol{m}$ might be much bigger than the object mask $\boldsymbol{m}_s$. Then, we use the image mask $\boldsymbol{m}$, to define $\boldsymbol{x}_{\text{BLANKED}} = \boldsymbol{x}_{\text{GT}} \odot \boldsymbol{m}$ and $\boldsymbol{s}_{\text{BLANKED}} = \boldsymbol{s}_{\text{GT}} \odot \boldsymbol{m}$.

To ease the description of our method, we split it in two parts: object generation and inpainting, which are described as follows.

3.2.1 Learning to generate instances

We aim at inserting new objects in the semantic space. Given an object $\boldsymbol{m}_s$, we want to learn a generative function starting from a latent code and a class generating the object $G_s(\boldsymbol{z}, \boldsymbol{c}) = \boldsymbol{m}_s$. As inserting a new object is an ambiguous task (e.g. pedestrians might have different poses or shapes), we want to learn to generate multiple plausible objects and results from which the user can choose from. Thus, we focus on learning a latent distribution of object shapes through an encoder network E_s, which will be then used to synthetize a new object shape in a VAE-fashion [56]. The VAE allows the generation of multiple objects by sampling multiple times from the learned latent distribution.

The shape and size of each object shape depends on its type (e.g. pedestrian) but also on its location in the scene. For example, pedestrians on the lower left part of the image are usually bigger than pedestrians on the upper part of the image, due to the different perspective. Thus, we formulate the problem as learning an encoder $\boldsymbol{z} \sim E_s(\boldsymbol{m}_s, \boldsymbol{c}, \boldsymbol{l})$, where $\boldsymbol{l}$ is the location vector of the shape.

Inspired by VAE literature [56], we assume a low-dimensional latent space distributed as a Gaussian, from which we can sample $\boldsymbol{z} \sim \mathcal{N}(0, \boldsymbol{I})$ where $\boldsymbol{I}$ is the identity matrix. Finally, we learn the encoder to learn the distribution with:

$$\mathcal{L}_s^{VAE} = -\mathcal{D}_{KL}(E_s(\boldsymbol{m}_s, \boldsymbol{c}, \boldsymbol{l}) \| \mathcal{N}(0, \boldsymbol{I})) \tag{3.1}$$

where $\mathcal{D}_{KL}(p\|q) = -\int p(t) \log \frac{p(t)}{q(t)} dt$ is the Kullback-Leibler divergence. This loss is ultimately expected to lead at learning the true posteriors of the latent distribution.

Then, we encourage the generated mask $\hat{\boldsymbol{m}}_s$ to be as similar as possible to the original one after the sampling.

$$\mathcal{L}_s^{rec} = \|\boldsymbol{m}_s - \hat{\boldsymbol{m}}_s\|_1 \tag{3.2}$$

We learn to generate realistic instances in an adversarial Least Square GAN [74], i.e. considering a loss:

$$\mathcal{L}_s^{adv}(G_s, D_s) = \frac{1}{2}\mathbb{E}_{\boldsymbol{m}_s}[(D_s(\boldsymbol{m}_s))^2] + \frac{1}{2}\mathbb{E}_{\hat{\boldsymbol{m}}_s}[(D_s(\hat{\boldsymbol{m}}_s) - 1)^2] \tag{3.3}$$

where D_s is the discriminator of object instances, based on DCGAN [89].

We jointly train the generator G_s and discriminator D_s using the following objective function:

$$\mathcal{L}_s^{adv}(G_s, D_s) = \lambda_s^{VAE}\mathcal{L}_s^{VAE} + \lambda_s^{rec}\mathcal{L}_s^{rec} + \lambda_s^{adv}\mathcal{L}_s^{adv} \tag{3.4}$$

3.2.2 Learning to inpaint

The goal of our model is not only to insert a new instance in the image, but also to modify the surrounding part by either altering the shape of existing objects, or removing them. Thus, we aim to complete the blanked image $\boldsymbol{x}_{\text{BLANKED}}$ and insert the object instance $\hat{\boldsymbol{m}}_s$ in the urban scene. As the task of inserting a new object operates on the semantic space, we facilitate the network by using the semantic segmentation, which is often available either through large scale human annotated datasets [14, 125, 81] or semi/weakly/un-supervised approaches [131, 64, 52].

Given a blanked image $\boldsymbol{x}_{\text{BLANKED}}$, its corresponding segmentation $\boldsymbol{s}_{\text{BLANKED}}$, and a desired object class $\boldsymbol{c}$, the network has to output an image $\boldsymbol{x}_{\text{FILLED}}$ and its corresponding segmentation $\boldsymbol{s}_{\text{FILLED}}$, which have the blanked part reconstructed and with the object instance inserted. Following Figure 3.2, we feed the network with the three inputs and we sample $\boldsymbol{z}$ from the latent distribution. E_{im} encodes the image $\boldsymbol{x}_{\text{BLANKED}}$, while E_{se} encodes both $\boldsymbol{s}_{\text{BLANKED}}$ and the generated instance $\hat{\boldsymbol{m}}_s$. We note that, since $\hat{\boldsymbol{m}}_s$ is a binary image, the network has to understand the object class from its shape. The intuition behind the two-streams encoder is that E_{im} focuses on learning the style of the image, while E_{se} focuses on the content and semantic by means of the semantic segmentation.

The two encoders are then jointly fused in the feature level through a composition of several Residual Blocks [37]. Then, in the decoder of the network, the image inpainting and semantic segmentation are progressively generated and updated across various decoder blocks with the segmentation map used to guide the inpainting of the masked image.

Segmentation reconstruction and decoder block. Motivated by the impressive results of the SPADE [86] normalization on image generation from semantic segmentation, we propose a new decoder block to guide the generation process. In the original formulation, SPADE exploits a human drawn semantic segmentation to synthetize photorealistic images. Here, we instead guide the normalization with a predicted segmentation map, thus without relying on human drawn semantic segmentations. This choice has two prominent advantages: i) it does not require the segmentation of the desired outcome (as we want to inpaint, we do not have it), and ii) it alleviates the mode collapse of the generated results that permits the nice consequence to have multiple diverse images.

Figure 3.2 (right) shows our decoder block. First, we get as input the features of the previous layer and upsample them using an upsampling sub-pixel convolution [104]. These upsampled features are used to predict the segmentation map $\boldsymbol{s}^l_{\text{FILLED}}$ and forwarded to the SPADE Resblock [86]. To remove any dependency of the batch we also replace SPADE Batch Normalization in favour of Instance Normalization [120].

To encourage consistency of real pixels and segmentation masks, but also of segmentation masks across the decoder layers, we use a multi-scale segmentation loss between the predicted segmentation mask and the ground truth

$$\mathcal{L}^{se}_c = -\sum_k^C \sum_{i=0}^m \boldsymbol{s}^k_{\text{GT}} \log(\mathit{upscale}(\boldsymbol{s}^k_i)) \tag{3.5}$$

where C is the number of considered semantics (e.g. road, car), and *upscale* interpolates $\boldsymbol{s}_i$ to match the size of $\boldsymbol{s}_{\text{GT}}$ and compute $\mathcal{L}^{se}_c$ on highly detailed segmentation maps.

Image reconstruction. We encourage the network at reconstructing the image with different state of the art losses:

- *Pixels reconstruction.* Given an image sampled from the data distribution, we should be able to reconstruct it after encoding and decoding. This can be obtained using:

  $$\mathcal{L}^{rec}_c = \|\boldsymbol{x}_{\text{FILLED}} - \boldsymbol{x}_{\text{GT}}\|_1 \tag{3.6}$$

  The $\mathcal{L}_1$ loss encourages the generation of sharper images than the $\mathcal{L}_2$ [47]. The loss is normalised by the mask size.

- *Feature-Matching Loss.* Inspired by [131], the feature-matching loss compares the activation maps of the layers of the discriminator. This forces the generator to produce images whose representations are similar to the ground truth in the discriminator space. It is defined as:

$$\mathcal{L}_c^{FM} = \mathbb{E}\Bigg[\sum_{k=1}^{K}\sum_{i=1}^{L} \frac{\|D_k^{(i)}(\boldsymbol{x}_{\text{GT}}^k, \boldsymbol{s}_{\text{GT}}^k) - D_k^{(i)}(\boldsymbol{x}_{\text{FILLED}}^k, \boldsymbol{s}_{\text{GT}}^k)\|_1}{K}\Bigg] \tag{3.7}$$

  where K is the number of scales of the discriminator, L is the last layer of the discriminator and $D_k^{(i)}$ is the activation map of the i_{th} layer of the discriminator.

- *Perceptual and Style Loss.* First introduced in [27, 51], $\mathcal{L}_c^{perc}$ is used to penalize results that are not perceptually similar to the source image, in this case the ground truth, and it is particularly important in complex scenes, where multiple details and objects are present. The perceptual distance is measured by the distance between the activation maps of the two images using a pretrained network. Formally:

$$\mathcal{L}_c^{perc} = \mathbb{E}\Bigg[\sum_{i=1}^{4} \frac{1}{N_i}\, \|\phi_l(\boldsymbol{x}_{\text{FILLED}}) - \phi_l(\boldsymbol{x}_{\text{GT}})\|_1\Bigg] \tag{3.8}$$

  where ϕ_i corresponds to the activation map of the i-th layer of a ImageNet pre-trained VGG-19 network [108]. $N_i = H_j W_j C_j$ is a normalization factor that takes into account the number of elements of VGG-19. The activation maps of the VGG-19 network are also used to compute the style loss, $\mathcal{L}_c^{style}$, which measures the differences between covariances of the activation maps and penalizes the style shifting of the two images. Formally:

$$\mathcal{L}_c^{sytle} = \mathbb{E}_j\Bigg[\Big\|(G_j^{\phi}(\boldsymbol{x}_{\text{FILLED}}) - G_j^{\phi}(\boldsymbol{x}_{\text{GT}}))\Big\|_1\Bigg] \tag{3.9}$$

  where G_j^{ϕ} is computed as $G_j^{\phi} = \psi\psi^T \frac{1}{N_i}$, ψ is ϕ_j reshaped into a $C_j \times H_j W_j$ matrix. Similarly to $\mathcal{L}_c^{perc}$, the style loss encourage high quality results in complex scenes.

We learn the generation through an adversarial with a Least Square loss [74]. We construct the adversarial game with a multiscale discriminator PatchGAN [131, 47]

$$\begin{aligned}\mathcal{L}_c^{adv}(G_s, D_s) = \sum_{k=1}^{K} & \frac{1}{2}\mathbb{E}_{\boldsymbol{x}_{\text{GT}}}[(D_g^k(\boldsymbol{x}_{\text{GT}}^k, \boldsymbol{s}_{\text{GT}}^k))^2]+ \\ & \frac{1}{2}\mathbb{E}_{\boldsymbol{x}_{\text{FILLED}}}[(D_g^k(\boldsymbol{x}_{\text{FILLED}}^k, \boldsymbol{s}_{\text{GT}}^k) - 1)^2]\end{aligned} \tag{3.10}$$

where K is the number of scales and k refers to the k^{th} scale of the image and segmentation map. Each layer k of the discriminator D_g takes as input the generated image $\boldsymbol{x}_{\text{FILLED}}^k$ and the ground truth segmentation map $\boldsymbol{s}_{\text{GT}}^k$. The ground truth segmentation map $\boldsymbol{s}_{\text{GT}}^k$ is here useful to verify the coherence between the generated image and the semantic segmentation.

The entire inpaint network is trained with the following losses:

$$\mathcal{L}_{D_c} = \mathcal{L}_c^{adv}(G_c, D_c) \tag{3.11}$$

$$\begin{aligned}\mathcal{L}_{G_c} =& \mathcal{L}_c^{adv}(G_c, D_c) + \lambda_c^{rec}\mathcal{L}_c^{rec}(G_c) + \lambda_c^{perc}\mathcal{L}_c^{perc}(G_c)+ \\ & \lambda_c^{style}\mathcal{L}_c^{style}(G_c) + \lambda_c^{FM}\mathcal{L}_c^{FM}(G_c) + \lambda_c^{cross}\mathcal{L}_c^{cross}(G_c)\end{aligned} \tag{3.12}$$

where λ_{rec}, λ_{perc}, λ_{style}, λ_{FM} and λ_{cross} are hyper-parameters for the weights of the corresponding loss terms. The value of most of these parameters come from the literature. We refer to Appendix B for the details.

3.3 Experiments

We conduct a quantitative evaluation on two widely recognized datasets for urban benchmarks, namely Cityscapes [14] and the Indian Driving [125]. The former is focused on European contexts while the latter exhibits a higher diversity of pedestrians and vehicles, but also ambiguous road boundaries and conditions. For both the datasets we first resize the images to height

256, then we random crop them to 256×256. Inspired by literature [14], we aggregate the 35 segmentation map categories into 17 groups for Cityscapes, and in Indian Driving we follow a similar approach from 40 categories to 21 groups. We refer to Appendix B for additional details.

3.3.1 Baselines

As baselines we select three state of the art models for image manipulation, namely Hong *et al* [40], and inpainting, i.e. SPG-Net [110] and RN [144]. Hong *et al* [40] is a two-stage manipulation model that uses the segmentation mask and insert or remove objects from an image. The user is required to specify the exact bounding box where to insert the object. SPG-Net is a two-stage inpainting network that leverages the semantic segmentation as input, while RN is a one stage inpainting network learning a normalization layer to achieve consistent performance improvements over the previous works. We used the source code released by RN authors, while we implemented from scratch SPG-Net following the description of the original paper as the code was not available. We release the code of our model and the implementation of SPG-Net.

3.3.2 Experimental settings

We design the following two testing setups for our network.

Restore. We test the models for image inpainting, where a portion of the image is blanked out and the network has to generate a plausible portion of the image.

Place. We test the models for the task of object insertion and image inpainting together. Thus, we blank out a portion of the image. The networks have to reconstruct the image and the desired object has to be present in the reconstruction. As SPG-Net and RN are not able to insert objects, we modify them to use an instance sampled from the latent space trained with the same VAE we use in our proposal. We call these networks SPG-Net* and RN*. Since Hong *et al* [40] is only able to place objects as big as the bounding box, we first restore the missing part, then we ask the network to place a new object in the exact position where the ground truth object is. We note that this setting might favour Hong *et al* [40] model.

In order to conduct a fair comparison with [110], for the *restore* task we use rectangular masks. Thus, for each image, we create a single rectangular mask at random locations. Each mask can have a size that goes from 32x32 up to 128x128. For the *place* task, we maintain the same size of the mask of the *restore* one but we change the position. In particular, we firstly extract a valid instance from the instance-wise annotation of each image. We pre-process the dataset extracting information for each object discarding instances that are too small or occluded from other objects. Then, we randomly choose one from the list of valid objects and we generate the mask based on the position of the instance. Whenever an image does not contain a valid object, we generate a new mask at random position. We release the generated mask to ease the comparison of future research with our model.

3.3.3 Evaluation

We evaluate generated results through image quality and instance accuracy. We measure image quality through the Peak Signal-to-Noise Ratio (PSNR) and the Fréchet Inception Distance (FID) [38]. The FID is the Wasserstein-2 distance between the generated and real image feature representations extracted from a pre-trained Inception-V3 model [115]. Higher PSNR and Lower FID values indicate higher image quality.

To measure the presence of inserted objects we use the F1 metric through

Table 3.1: Quantitative results for our model and the baselines. We evaluate the models through image quality (PSNR and FID) and accuracy of instance insertion (F1).

	Model	**Cityscapes**			**Indian Driving**		
		PSNR↑	FID↓	F1↑	PSNR↑	FID↓	F1↑
Restore	Hong *et al.* [40]	31.07	7.26	0.00	30.31	6.34	0.00
Restore	SPG-Net [110]	31.36	7.97	0.00	29.95	6.39	0.02
Restore	RN [144]	32.16	9.64	0.00	29.83	11.14	0.02
Restore	Our proposal	**32.95**	**5.08**	**0.06**	**30.97**	**5.45**	**0.05**
Place	Hong *et al.* [40]	31.08	7.26	0.10	30.32	6.32	0.91
Place	SPG-Net*	31.37	7.96	0.60	29.94	6.38	0.87
Place	RN*	31.74	9.79	0.54	29.62	10.76	0.71
Place	Our proposal	**32.96**	**5.05**	**0.91**	**30.98**	**5.43**	**0.97**

the use of a pre-trained YOLOv3 [92] network, which detects whether the desired instances are inserted. Higher scores indicate that the network is inserting synthetic objects that resemble the real ones. For the *restore* experimental setting, we compute the score by detecting whether at least one of the modelled instances (i.e. cars and pedestrians) is inserted.

Perceptual user study. We also run a perceptual study asking 19 users to conduct a user study on both datasets. Each user evaluates 38 random images, 19 for each experimental setting. For each image we show the original masked image and the generated output of all the networks. The users are asked to select the best output judging the realism and quality of reconstructed pixels.

3.4 Results

Quantitative results. We evaluate our proposal with the state of the art solutions in two settings: *restore* and *place*, which test the ability to inpaint missing portions of images and manipulate an image (insert, remove,

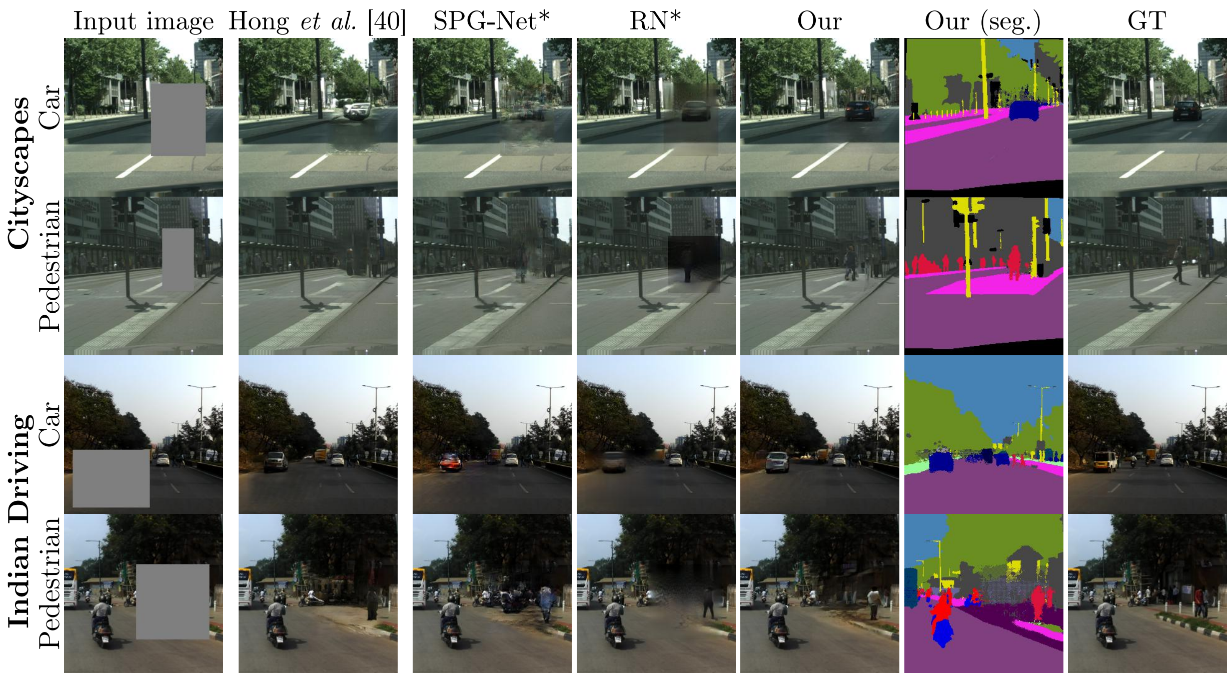

Figure 3.3: Qualitative evaluation on the task of object insertion and inpainting. The first two rows show results on the Cityscape dataset while the last two rows show the Indian Driving results. We show the results on the same multi-domain model conditioned on two types of object insertion: cars and pedestrians.

reconstruct), respectively. Table 3.1 shows that our approach outperforms all the compared models in the two settings. In particular, we observe that our model significantly improves the quality of generated images, measured through FID and PSNR. In the *restore* setting, we observe that, as expected, all inpaiting models collapse to object removal, rarely proposing the insertion of a new object in an existing scene. Thus, we condition existing models to insert new objects (*place* setting). Without significant losses on the image quality, we observe that both SPG-Net* and RN* learn to insert object instances. However, our proposal greatly improves both the image quality and the accuracy of object insertion, which increases on average by 18% and 310% in Indian Driving and Cityscapes, respectively. Overall, we also show that merely changing existing models (RN and SPG-Net) does not achieve state of the art performance.

Qualitative results. We begin by commenting on the results of the most challenging *place* experimental setting. Figure 3.3 shows four different images from which we want to reconstruct the missing part but also insert a user-defined object (e.g. car or pedestrian). We can see that our method correctly inserts the required object, while others struggle at doing it, especially in Cityscapes. While RN is often able to generate object shapes that are correctly detected by YOLOv3 [92], a visual inspection of the second and fourth row of Figure 3.3 highlights that the object is poorly colorized. Moreover, all the baselines fail at adapting the reconstructed pixel to the scene. In particular, SPG-Net and Hong *et al.* suffer from significant blurriness and artifacts (see Figure 3.3 first rows), while RN unreliably reconstructs the edges and result in blurry reconstructions. On the contrary, our proposal results in sharper images and distinguishable inserted objects.

We now discuss the *restore* results where the model has only to reconstruct missing parts of the image. Figure 3.4 shows that our model generates sharper images with pixels that are well adapted in the original scene. In particular, we observe that our model can reconstruct even the horizontal road marking quite well (see the first row Figure 3.4). State of the art models, instead, seem to struggle at the reconstruction. These poor results might be a consequence of two main reasons. First, RN focuses on free-form missing parts, while we might generate big areas where their normalization might fail. Second, the reason behind the "checkerboard" artifacts of SPG-Net might lie on the Deconvolution applied in the decoder in order to upsample the features. Indeed, Deconvolution has been shown to produce results that present various artifacts [83] and that can be reduced using style loss [97].

Thus, we propose a new decoder block, based on SPADE, that jointly predicts the segmentation and real pixels. By using the predicted segmentation as input to the SPADE normalization, we guide the decoder to reconstruct semantics that are consistent with the real pixels. As a result, edges and contours are visually pleasing and better reconstructed than the state of the art models. It is worth noting that our decoder can also be applied easily to other existing models.

Our holistic framework can be applied in a wide range of manipulations of complex scenes. Thus, we also test our model performance in inserting and removing an user specified input shape (see Figure 3.1 for the manipulation types). Qualitative and quantitative results for these two tasks show that our model significantly outperforms the baselines (see Appendix B Figure B.1 and Figure B.2). Surprisingly, we qualitatively observe that RN generates low quality results and visible boundaries even in the free-form task in which they focus, highlighting the challenge of complex scene manipulation.

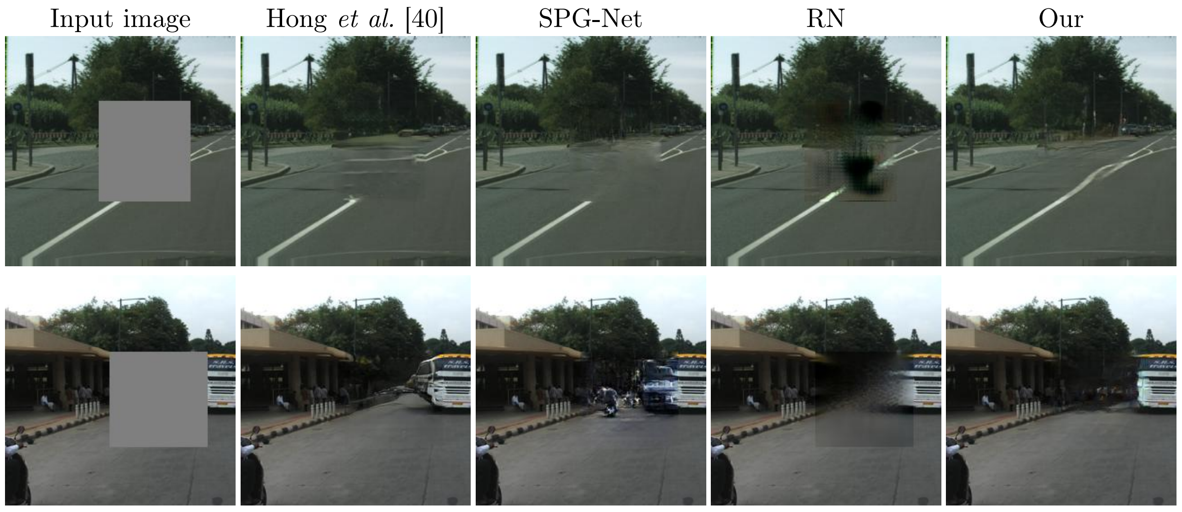

Figure 3.4: Qualitative evaluation on the task of inpainting without object insertion. The first row shows results on the Cityscape dataset, while the last row shows the Indian Driving results. Zoom in for better details.

Perceptual user study. The user study shows that our generated images have been selected 85% of the time. Specifically, our results are perceptually better 90% and 80% of the time in the *restore* task and *place* tasks respectively.

Ablation. Our model introduces various novel components and different state of the art losses whose contribution has to be validated. We performed

Table 3.2: Ablation study of the architecture and input quality.

Model	**PSNR↑**	**FID↓**
Our proposal (A)	**32.96**	**5.05**
(A) w/o $\mathcal{L}_{\text{style}}$	32.68	5.38
(A) w/o $\mathcal{L}_{\text{FM}}$	32.66	5.30
(A) w/o E_{se} and $\boldsymbol{s}_{\text{BLANKED}}$	32.35	5.42
(A) w/o SPADE	32.57	5.56
(A) using DeepLabv3 [9] segmentation	32.85	5.11

an ablation study on Cityscapes, which is the standard testbed for inpainting in complex scenes.

We begin by testing the contribution of the style $\mathcal{L}_{\text{style}}$ and feature matching $\mathcal{L}_{\text{FM}}$ losses. Table 3.2 shows that removing one of the two losses results in a small performance drop. However, qualitative results show they reduce the "checkerboard" artifacts in the image, which is confirmed by recent literature [97] (see Appendix B).

Similar to SPG-Net, our network uses a segmentation mask in input, which is supposed to guide the network at learning a better feature representation. However, it is arguably a costly choice as it requires human annotations. Moreover, our decoder block predicts the segmentation and uses the SPADE normalization, which might be sufficient to have high-quality results. To test these two hypotheses, we first train a network without the segmentation encoder E_{se} and the segmentation map $\boldsymbol{s}_{\text{BLANKED}}$. From Table 3.2, we can observe a significant drop in performance, but our network still performs better than state of the art without these two components. Thus, this setting might be considered in limited-resource scenarios. Then, we test the use of predicted segmentation maps instead of the ground truth maps. So, we use segmentation images generated by a pre-trained segmentation predictor DeepLabv3 [9]. We observe no significant performance degradation.

Finally, we observe that by removing the entire SPADE block we see a significant negative effect on image quality. However, these results are better than the state of the art models, which show that the contribution of our model is not only a consequence of SPADE. We refer to Appendix B for additional ablations for components of our SPADE block.

3.5 Conclusion

In this chapter, we proposed a novel framework that unifies and improve previous methods for image inpainting and object insertion. At inference time, users interact with our model by feeding as input an image and its segmentation that have blanked area, and optionally an object (label or shape) to be inserted. The model encodes both the two input images, samples from a latent distribution whenever a class label is fed, and reconstructs the images. If users desire to insert the object, the reconstructed images contain it as expected.

To encourage high quality reconstructions, we use semantic segmentation maps both in input and in the decoder. Specifically, we proposed a new decoder block that is based on SPADE and mixes a semantic prediction task with the normalization to generate adequate images. We believe that this decoder block might be applied to a wide range of tasks that exploit the semantic segmentation to generate new images.

To the best of our knowledge, we are the first at learning to jointly remove, inpaint and insert objects of multiple types in a single one-stage model. We hope that our work will stimulate future research on the challenging manipulation of complex scenes, having multiple cluttered objects and semantic classes.

Chapter 4

Controllable Video Generation

Recent years have witnessed several breakthroughs in the generation of high dimensional data such as images [12, 20, 72] or videos [119, 129]. However, most practical and commercial applications require to control generated visual data on inputs provided by the user. For instance, in image manipulation, photo editing software [45] applies deep learning models to allow users to change portions of an image [144, 102, 80].

Regarding videos, several possible ways to control the generated sequences have been considered. For instance, the generation of frames can be conditioned on simple categorical attributes [36], short sentences [66] or sound [118]. An interesting recent research direction comprises works that attempt to condition the video generation process providing motion information as input [119, 133, 106, 107]. These approaches allow to generate videos of moving faces [133], human silhouettes and, in general, of arbitrary objects [119, 106, 107]. However, these works mainly deal with videos depicting a single object. It is indeed extremely more challenging to animate images and generate videos when multiple objects are present in the scene, as there is no simple way to disentangle the information associated with each object and easily model and control its movement.

This chapter introduces Click to Move (C2M), the first approach that allows

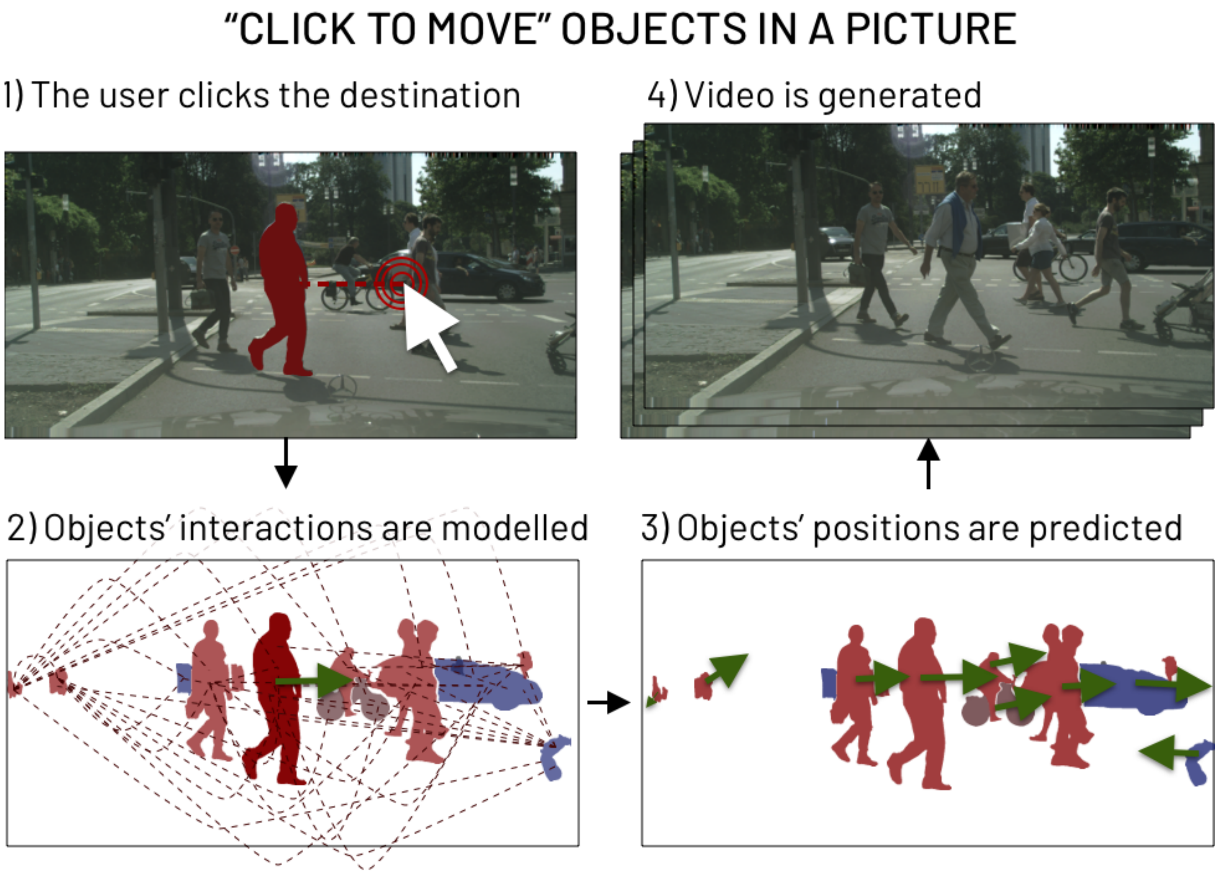
"CLICK TO MOVE" OBJECTS IN A PICTURE
1) The user clicks the destination
4) Video is generated
2) Objects' interactions are modelled
3) Objects' positions are predicted

Figure 4.1: Illustration of the video generation process of Click to Move (C2M): 1) the user selects the objects in a scene and specify their movements. 2) Our network models the interactions between *all* objects through the GCN and 3) predicts their displacement. 4) The network produces a realistic and temporally consistent video.

users to generate videos in complex scenes by conditioning the movements of specific objects through mouse clicks. Fig.4.1 illustrates the video generation process of C2M. The user only needs to select few objects in the scene and to specify the 2D location where each object should move. Our proposed framework receives as inputs an initial frame with its segmentation map and synthesizes a video sequence depicting objects for which movements are coherent with the user inputs. The proposed deep architecture comprises three main modules: (i) an appearance encoder that extracts the feature representation from the first frame and the associated segmentation map, (ii) a motion module that predicts motion information from user inputs and image features, and (iii) a generation module that outputs the synthesised frame sequence. In complex scenes with multiple objects, modelling interactions is essential to generate coherent videos. To this aim, we propose to adopt a Graph Neural Network (GCN), which models object interactions and infers the plausible displacements for all the objects in the video, while respecting the user's constraints. Experimental results show that our approach outperforms previous video generation methods on two publicly available datasets and demonstrate the effectiveness of the proposed GCN framework in modelling object interactions in complex scenes.

Our work is inspired by previous literature that generates videos from an initial frame and the associated segmentation maps [85, 103]. From these works, we inherit a two-stage procedure where we first estimate the optical flows between an initial frame and all the generated frames, and subsequently refine the image obtained by warping the initial frame according to the estimated optical flows. However, our framework improves over these previous works as it allows the user the possibility to directly control the video generation process with simple mouse clicks. Similarly to the work of Hao *et al.* [35], we propose to control object movements via sparse motion inputs. However, thanks to the GCN, our approach can deal with scenes with multiple objects, while [35] cannot. Furthermore, the method in [35] does not explicitly consider the notion of *object*, as it does not use any instance segmentation information, and does not model the temporal relation between multiple frames. We instead work on multiple frames and in the semantic space, so the user can intuitively select the object of interest and move it in a temporal consistent way. The use of semantic information is motivated by recent findings in the area of image manipulation where it has been shown that semantic maps are beneficial in complex scenes [1, 62].

Overall, the main contributions of our work are as follows:

- We propose Click to Move (C2M), a novel approach for video generation of complex scenes that permits user interaction by selecting objects in the scene and specifying their final location through mouse clicks.
- We introduce a novel deep architecture that leverages the initial video frame and its associated segmentation map to compute the motion representations that enable the generation of frame sequence. Our deep network incorporates a novel GCN that models the interaction between objects to infer the motion of all the objects in the scene.
- Through an extensive experimental evaluation, we demonstrate that the proposed approach outperforms its competitors [85, 103] in term of video quality metrics and can synthesize videos where object movements follow the user inputs.

4.1 Related Works

Video generation with user control. With the recent progress in deep video synthesis, researchers have focused in designing new approaches that include user input in the generation process. Video generation can be controlled by different means. For example, MoCoGAN [119] disentangles videos into motion and content latent spaces. Therefore, it is possible to control videos by “copying" the action from another video or by changing the identity of the person. Chan *et al.* [6] propose to generate dance videos following a “do as I do" motion transfer strategy: body poses are estimated for every frame of another video and transferred to control the pose of the person in the generated video. Wiles *et al.* [133] control human face motion through a driving vector that can be extracted from videos or pose information. Siarohin *et al.* [106, 107] propose an approach suitable to arbitrary objects and learn motion representations without requiring specific prior knowledge. This approach can be employed with various types of videos, ranging from human bodies to robotics. Regarding audio-visual methods, talking heads video can be generated from an initial image and an input audio clip [133, 8, 151]. In this chapter, we propose a novel framework that involves the user in the generation process. However, while previous works mostly focus on generating videos depicting a single object (e.g. a face or human body), we address the more challenging task of video synthesis of complex scenes where multiple objects have to move consistently while accounting for user input.

Future frame prediction. The problem we address in this work is closely related to future frame prediction, which aims to generate a video sequence given its initial frames. Early works formulate the problem as a deterministic prediction task [23, 77, 128]. However, this formulation cannot work on most real world videos due to the inherent motion uncertainty. Thus, recent approaches adopt adversarial [59] or variational [25, 61, 119] formulations that can model stochasticity. Several works focus on the architectural design and propose to estimate optical flow [59, 24, 67, 65] to generate the future frames by warping the previous one. Other works study solutions for long term predictions [126, 39, 141, 91]. Similarly, Li *et al.* [65] propose a multi-step network that first generates an optical flow, then converts it back to the RGB space to generate novel videos. Instead, Zhang *et al.* [148] propose to employ an optical flow encoder that maps motion information to a latent space. At test time, different random motion vectors can be sampled to generate video with different motion.

When it comes to complex environment involving multiple objects, additional supervision is highly beneficial. For example, Wu *et al.* [134] use video frames, optical flows, instance maps and semantic information together to decouple the background from the dynamic objects and thus predict their trajectory. Similarly, Hao *et al.* [35] show that providing sparse motion trajectories to their model helps generating videos with higher quality. However, contrary to our approach, their method does not take advantage of instance segmentation and does not model object interactions.

Recently, Pan *et al.* [85] and Sheng *et al.* [103] have proposed to get a benefit from segmentation information to improve video generation. Videos are generated from a single frame and the corresponding segmentation map. Both approaches are based on a two-stage procedure. The first stage aims at estimating the optical flow between the initial frame and every generated frame. In the the second stage, the initial frame is warped according to the optical flow and refined by an encoder-decoder network. Inspired from these works, our approach adopts a similar variational auto-encoder framework boosted with optical flow and occlusion supervision. However, we include a novel Graph Convolutional Network (GCN) that models object interactions and takes into account the sparse motion vectors provided by the user.

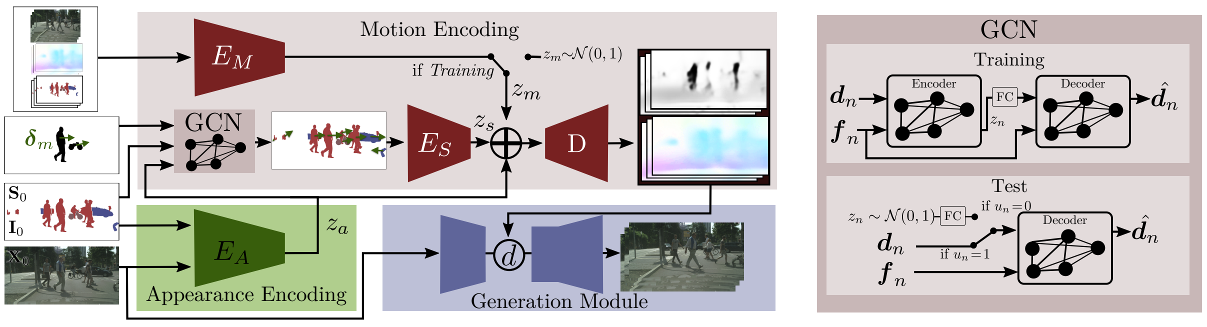

Figure 4.2: Our network is composed of three modules, namely (i) Appearance encoding, (ii) Motion encoding, and (iii) Generation module. The Appearance Encoding focuses on learning the visual appearance from $\mathbf{X}_0$. The Motion Encoding models the interactions between the objects, predicts their displacement, encodes the motion, and generates the optical flow and occlusion mask for the Generation Module, which focuses on generating temporal consistent and realistic videos. On the right, we show our GCN module to model objects' interactions.

4.2 Click to Move framework

We aim at generating a video from its initial frame $\mathbf{X}_0 \in \mathbb{R}^{H\times W\times 3}$ and a set of user-provided 2D vectors that specify the motion of the key objects in the scene. At test time, we assume that we also have at our disposal the instance segmentation maps of the initial frame. Our system is trained on a dataset of videos composed of T frames with the corresponding instance segmentation maps at every frame. As we will see later, in practice, instance segmentation is obtained using a pre-trained model.

Considering a set of C classes, we assume that N objects are detected at time t in the frame $\mathbf{X}_t \in \mathbb{R}^{H\times W\times 3}$. The instance segmentation is represented via a segmentation map $\mathbf{S}_t \in \{0,1\}^{H\times W\times C}$, a class label map $\mathbf{C}_t \in \{1,...,C\}^{H\times W}$ and an instance map $\mathbf{I}_t \in \{1,...,N\}^{H\times W}$ that specifies the instance index for every pixel. At test time, the user provides the motion of the M objects in the scene by drawing 2D arrows corresponding to the displacement between the barycenter of the object in $\mathbf{X}_0$ and the object's desired position at time T (See Fig. 4.1). Notably, the user is free to provide motion vectors for as many objects as desired. Therefore the motion vectors are represented by a list $\mathcal{M} = \{(\boldsymbol{\delta}_m, i_m), 1 \leq m \leq M\}$, where $\boldsymbol{\delta}_m \in \mathbb{R}^2$ contains the barycenter displacement of the object with instance index i_m. At training time, the list $\mathcal{M}$ is obtained by randomly sampling objects in every video and estimating their corresponding $\boldsymbol{\delta}_m$, which is defined as the displacement of the instance segmentation's barycenters between the first and last frame.

The proposed framework is articulated in three main modules, as illustrated in Fig. 4.2. First, the *Appearance encoding* is in charge of encoding the initial frame. This module receives as input the concatenation of the initial frame $\mathbf{X}_0$, the segmentation $\mathbf{S}_0$ and the instance map $\mathbf{I}_0$, while it outputs a feature map $\boldsymbol{z}_a$ via the use of an Encoder E_A. Second, the *Motion encoding*, predicts the video motion from the motion vectors provided by the user and the image features $\boldsymbol{z}_a$. This module includes a novel Graph Convolutional Network (GCN) that infers the motion of all the objects in the scene by combining the object motion vectors in $\mathcal{M}$ and the image features $\boldsymbol{z}_a$. This motion module is described in Sec. 4.2.2 while the details specific to our GCN are given in Sec. 4.2.1. Finally, the *Generation module* is in charge of combining the encoded appearance and the predicted motion to generate every frame of the output video.

4.2.1 Object motion estimation with GCNs

Our GCN aims at inferring the motion of all the objects in the scene by combining the motion vectors provided by the user and the image features $\boldsymbol{z}_a$. This section first describes the specific message-passing algorithm that we introduce to model the motion vectors. Then we show how our GCN is embedded into a Variational Auto-Encoder (VAE) framework to allow sampling the possible object motions that respect the user's constraints.

Handling user control with GCNs. We propose to use a graph to model the interactions between the objects in the scene. Each node corresponds to one of the N objects detected in $\mathbf{X}_0$. The graph is obtained fully connecting all the objects with each other. Let us introduce the following notations: $\boldsymbol{f}_n$ is the feature vector for the n^{th} object and is extracted from $\boldsymbol{z}_a$ via region-wise average pooling. $\boldsymbol{d}_n \in \mathbb{R}^2$ is the estimated barycenter displacement for the n^{th} object. Finally, $u_n \in \{0, 1\}$ is a binary value that specifies whether the object motion has been provided by the user ($u_n = 1$) or if it should be inferred ($u_n = 0$).

In a standard GCN [136], the layer-wise propagation rule specifies how the features $\boldsymbol{f}_n^{(k)}$ at iteration k of the node n are computed from the features of it neighbouring nodes at the previous iteration $\boldsymbol{f}_j^{(k-1)}$:

$$\boldsymbol{f}_n^{(k)} = \sum_{j \in \mathbf{N}(n)} \frac{1}{\sqrt{\mathcal{D}_{nj}}} \boldsymbol{\theta}^\top \boldsymbol{f}_n^{(k-1)} \tag{4.1}$$

where $\mathbf{N}(n)$ denotes the neighbours of the node n, $\boldsymbol{\theta}$ are the trainable parameters and $\mathcal{D}_{nj}$ is a normalization factor equal to the sum of the degree of the nodes n and j. In our context, we need to modify this update rule to take into account that the object motion of each node is either known or unknown. Besides, we propose two different propagation rules for the node features $\boldsymbol{f}_n$ and the motion vectors $\boldsymbol{d}_n$. We propose to make these rules depending on u_n. If $u_n = 1$, the node corresponds to an object with a motion controlled by the user and we update only the features:

$$\boldsymbol{f}_n^{(k)} = \boldsymbol{f}_n^{(k-1)} + \sum_{j \in \mathbf{N}(n)} \frac{1}{\sqrt{\mathcal{D}_{nj}}} \boldsymbol{\theta}_f^\top (\boldsymbol{f}_n^{(k-1)} \oplus \boldsymbol{d}_n^{(k-1)}) \tag{4.2}$$

$$\boldsymbol{d}_n^{(k)} = \boldsymbol{d}_n^{(k-1)}. \tag{4.3}$$

Here, $\boldsymbol{\theta}_f$ denotes the trainable parameters and $\oplus$ is the concatenation operation. This formulation allows propagating feature information through the node while keeping the object motion constant for the nodes with known motion. Note that, in equation 4.2, we opt for a residual update since the messages from the neighbouring nodes are added to the current value $\boldsymbol{f}_n^{(k-1)}$. Our preliminary results showed that equation 4.1 update rule ended up with all the nodes having the exact same features. On the contrary, the residual update that helped objects converging to better features. Indeed, this residual update can be seen as skip connections, similar to those of resnet architectures, that allow gradient information to pass through the GCN updates and mitigate vanishing gradient problems.

If $u_n = 0$, the node corresponds to an object with unknown motion and we update both the features and the motion vector. The feature update remains identical to equation 4.2 and the motion vector is updated as follows:

$$\boldsymbol{d}_n^{(k)} = \boldsymbol{d}_n^{(k-1)} + \sum_{j \in \mathbf{N}(n)} \frac{1}{\sqrt{\mathcal{D}_{nj}}} \boldsymbol{\theta}_d^\top (\boldsymbol{f}_n^{(k-1)} \oplus \boldsymbol{d}_n^{(k-1)}) \tag{4.4}$$

where $\boldsymbol{\theta}_d$ denotes the trainable parameters for the motion estimation. This novel propagation rule allows to aggregate the information contained in the neighbouring nodes to refine the motion estimation of nodes with unknown motion. In the next section, we detail how this GCN is embedded into a VAE framework in order to sample possible object motions.

Overall architecture for motion sampling. Our GCN is embedded into a VAE framework composed of an encoder and a decoder network. At training time, we employ an encoder and a decoder while only the decoder is used at test time, as illustrated in Fig. 4.2-Right. Note that the features $\boldsymbol{f}_n$ condition both the encoder and the decoder. The goal of the encoder network is map the input value $\boldsymbol{d}_n$ of every node to a latent space $\boldsymbol{z}_n$. This encoder is implemented using a GCN that employs the propagation rule described in Sec 4.2.1 and receives as input $\boldsymbol{f}_n \oplus \boldsymbol{d}_n$ for every node. For every node, the latent variable $\boldsymbol{z}_n$ is given by $\boldsymbol{f}_n^{(k)}$ after the last message propagation update. We assume $\boldsymbol{z}_n$ follows a unit Gaussian distribution ($\boldsymbol{z}_n \sim \mathcal{N}(0,1)$). The decoder network receives as input the randomly sampled latent variable $\boldsymbol{z}_n$ for the nodes with unknown motion (*i.e.* $u_n = 0$) and is trained to reconstruct the input motion $\boldsymbol{d}_n$. The decoder is implemented with another GCN with the same propagation rules and with inputs $\boldsymbol{f}_n^{(0)} \oplus \boldsymbol{d}_n^{(0)}$ where $\boldsymbol{f}_n^{(0)} = \boldsymbol{f}_n$ and:

$$\boldsymbol{d}_n^{(0)} = \begin{cases} \mathrm{FC}(\boldsymbol{z}_n) & \text{if } u_n = 0 \\ \sum_{m=1}^{M} \mathbb{1}(i_m = n)\boldsymbol{\delta}_m & \text{if } u_n = 1. \end{cases} \tag{4.5}$$

where $\mathbb{1}$ denotes the indicator function and FC(.) denotes a fully-connected layer that projects the sampled latent variable $\boldsymbol{z}_n$ to the space of $\boldsymbol{d}_n$ (*i.e.* $\mathbb{R}^2$).

Intuitively, the sum in equation 4.5 iterates over all the objects in $\mathcal{M}$ to select the corresponding motion vector provided by the user.

At test time, the GCN encoder is not used. The latent variable $\boldsymbol{z}_n$ is sampled according to our unit Gaussian prior distribution for every object with unknown motion and forwarded to the decoder. The decoder outputs the 2D motion of every object in the scene.

4.2.2 Motion encoding

This module is in charge of predicting the optical flows and the occlusion maps between the initial frame $\mathbf{X}_0$ and every frame that has to be generated. To this aim, for every time step t, we compute a binary tensor $\mathbf{B}_t \in \{0,1\}^{H\times W}$ that specifies the locations of the objects in the scene. At time $t=0$, the object-location map $\mathbf{B}_0$ is computed from the instance segmentation map $\mathbf{I}_0$:

$$\forall (i,j) \in H \times W, \mathbf{B}_0[i,j] = \sum_{n}^{N} \mathbb{1}(\mathbf{I}_0[i,j] = n). \tag{4.6}$$

For $t > 0$, $\mathbf{B}_t$ cannot be estimated with the previous equation since $\mathbf{I}_t$ is not known at test time. Instead, we consider a simple rigid model for every object and obtain $\mathbf{B}_t$ by warping $\mathbf{B}_0$ according to the the object motion $\boldsymbol{d}_t$. At training time, $\boldsymbol{d}_t$ is estimated from the segmentation maps while, at test time, we employ $\hat{\boldsymbol{d}}_t$, which is the displacement predicted by our GCN. Finally, this object-location tensor is mapped to a latent tensor $\boldsymbol{z}_s$ via an encoder E_S.

Note that, the output video cannot be fully encoded via the initial frame and the motion of each object since there exist other sources of variability such as the appearance of new objects or change in object sizes. Therefore, we introduce a latent motion variable $\boldsymbol{z}_m$ that encodes all the motion information that cannot be described by $\boldsymbol{z}_s$ and $\boldsymbol{z}_a$. We employ an auto-encoder strategy at training time, estimating $\boldsymbol{z}_m$ from the complete video sequence with an encoder E_M. More precisely, E_M receives as input the concatenation of all the video frames, the instance segmentation maps S_0 and I_0, and the optical flow for every frame. At test time, the latent motion code $\boldsymbol{z}_m$ is sampled according to the prior distribution ($i.e.$ $\boldsymbol{z}_m \sim \mathcal{N}(0, I)$).

Finally, we provide the latent variables $\boldsymbol{z}_a$, $\boldsymbol{z}_s$ and $\boldsymbol{z}_m$ to the same decoder, which outputs the bi-directional optical flows and occlusion maps. More precisely, the decoder outputs the forward and the backward optical flow at every time steps denoted by $\mathbf{F}_t^f$ and $\mathbf{F}_t^b$ respectively and the corresponding occlusion maps $\mathbf{O}_t^f$ and $\mathbf{O}_t^b$. Note that the backward optical flows and occlusion maps are then provided to the generation modules, while the forward optical flow and occlusion maps are used only for loss computation.

4.2.3 Generation module and training objectives

We employ a generation module inspired by [107]. After two down-sampling convolutional blocks applied on the initial frame $\mathbf{X}_0$, we obtain a feature map. We proceed independently for every frame to generate and warp the feature map according to the optical flow predicted by the motion module. Then we multiply the warped feature map by the occlusion map predicted by the occlusion estimator to diminish the impact of the features corresponding to the occluded parts. Finally, the masked feature maps are fed to a subsequent network to output the generated video. This network is composed of several residual blocks, followed by two up-sampling convolutional blocks.

Objective functions. Our GCN framework employs the evidence lower bound of the VAE framework. It is composed of a reconstruction term on the predicted motion vector and the Kullback-Leibler divergence (KL) between the conditional distribution of $\boldsymbol{z}_n$ and its unit Gaussian prior:

$$\mathcal{L}_{VAE} = \frac{1}{N} \sum_{n=0}^{N} \|\boldsymbol{d}_n - \hat{\boldsymbol{d}}_n\|_1 - \mathcal{D}_{KL}(\boldsymbol{z}_n \| \mathcal{N}(0, I)), \tag{4.7}$$

where $\hat{\boldsymbol{d}}_n$ is the displacement predicted by the GCN.

Forward-backward Consistency. Similarly to [102], we ensure the cycle consistency between forward and backward optical flows. More precisely, for every non-occluded pixel location $\boldsymbol{p}$, we minimize the L_1 distance between the corresponding optical flows:

$$\begin{aligned}\mathcal{L}_{Fc}(F^f, F^b) = \frac{1}{T}\sum_{i=1}^{T}\sum_{\boldsymbol{p}} \mathbf{O}_t^f(\boldsymbol{p})|\mathbf{F}_t^f(\boldsymbol{p}) - \mathbf{F}_t^b(\boldsymbol{p} + \mathbf{F}_t^f(\boldsymbol{p}))|_1 \\ + \mathbf{O}_t^b(\boldsymbol{p})|\mathbf{F}_t^b(\boldsymbol{p}) - \mathbf{F}_t^f(\boldsymbol{p} + \mathbf{F}_t^b(\boldsymbol{p}))|_1\end{aligned} \tag{4.8}$$

Smoothness. Following [103], we employ a smoothness loss that penalizes high gradient values in the optical-flow map that do not correspond to high-gradient values in the image $\mathbf{X}_0$ (for more details refer to [103]).

Supervised flow. To improve the quality of the generated videos in our multi-objects setting, we take advantage of a pre-trained FlowNet2 [44] network for optical flow and occlusion estimation. FlowNet2 provides high quality optical flow maps that we use as supervision for our motion decoder network using a standard L1 loss.

Motion Encoding uncertainty. To allow the sampling of $\boldsymbol{z}_m$ at test time, the output of the motion encoder E_M is mapped to a unit Gaussian distribution via the KL-divergence:

$$\mathcal{L}_m = -\mathcal{D}_{KL}(z_m \| \mathcal{N}(0, I)) \tag{4.9}$$

Generation module. The generation module is trained using state-of-the-art losses for video generation. Following [74, 131, 47] we adopt a PatchGAN discriminator trained with a Least Square loss. For the generator, we apply the structural similarity loss [132], the perceptual loss [51], feature matching loss [131], and a standard pixel-level reconstruction L1 loss.

4.3 Experiments

Datasets. We evaluate our model with two publicly available datasets, namely Cityscapes and KITTI 360.

- *Cityscapes* [14] provides videos at 17 Frames Per Second (FPS) of European urban scenes. We resize all images to 256×128 resolution for performance reasons. The dataset contains 2975 video sequences for training and 500 video sequences for testing. Since Cityscapes does not provides instance and semantic segmentations for the video sequences, we used [11] to generate them.
- *KITTI 360* [137] provides a richly annotated videos at 11 FPS in German suburban areas. We resize all images to 192×64 resolution. The dataset for our evaluation contains 6941 training videos and 423 test sequences. We aggregate the segmentation categories to match the 19 classes of Cityscapes.

Baselines. We compare with the state-of-the-art model for video generation in complex scenarios, *i.e.* Sheng *et al.* [103], which can generate high-quality videos from a staring frame and its associated semantic segmentation map. Since Sheng *et al.* [103] is not able to generate videos controlling object positions, we modify it by including the object location tensor $\mathbf{B}_t$ into the appearance encoder of the original model. We call this model *Sheng**. For a fair comparison, we also test our approach with a variant of the method of Sheng *et al.*, referred to as *S. Sheng**, where we add our *Supervised flow* loss that uses the supervision of a pretrained network in order to improve optical flow prediction. We note that Sheng *et al.* [103] is an extension of Pan *et al.* [85] and that these two works correspond to the same method. Thus, Pan *et al.* [85] is not included in our comparison. It is also worth that Hao *et al.* [35] is not included in the baselines, as it focuses on image generation and does not *explicitly* model the semantic space. Thus, it would be unfair to compare Hao *et al.* with our method on the temporal consistency and object displacements in videos.

Settings. We design three test settings to evaluate our proposal extensively.

- *Oracle (O).* For each video, we select a random object that has to be moved, we feed the networks with the ground truth displacements between the first and last frames, and let the models generate the video. This setting evaluates the network capacity to benefit from the given sparse motion information.
- *Custom.* For each input video, we select a random object that has to be moved, we feed the networks with displacement shifted by $\lambda = 1.5$ (i.e. $\boldsymbol{d}'_n = \lambda \boldsymbol{d}_n$) and let the models generate the video. This setting evaluates the network capacity to condition the video on sparse motion inputs, which are different from the ground truth.

Then, we also experiment a drastic scenario where *all* the objects are moved following the *Custom*. In this experiments, all future positions are provided as input. In this experiment, the GCN can be by-passed since $u_n = 1$ for every object. This experiments differ from *Ground truth* and *Custom* where our GCN has to infer the plausible future positions of all the objects that are not provided by the user. In all our experiments, we generate 5 future frames starting from the provided initial frame.

Evaluation metrics.

- *FVD.* We adopt the Fréchet video distance (FVD) metric [122] to evaluate both the video quality and temporal consistency of generated frames. We compute the FVD between the ground truth test videos and the generated ones. The lower the FVD, the better.
- *NDE.* We measure the adherence of generated videos with the user-provided motions by computing the Normalised Displacement Error (NDE) as the Euclidean distance between the coordinate specified by the user and the coordinate where the object ends-up in the generated video, which is then

normalised Euclidean distance of the ground truth starting coordinate and the ending one. All object's positions are detected through YOLOv3 [92]. We discard the objects that cannot be detected in the ground truth videos due to the resolution of videos, or because objects are too small to be correctly detected by YOLOv3. The lower the NDE, the better.

- *Acc.* The object's positions in generated videos can be difficult to track due to the presence of artifacts, occlusions and low-quality images. Thus, we report here the Accuracy (Acc) of the YOLOv3 detector in generated videos. The higher the Accuracy, the better.

Model	**FVD**$\downarrow$	**NDE**$\downarrow$	**Acc**$\uparrow$
A: Our proposal	288	1.01	0.84
B: (A) w/o GCN	369	1.42	0.70
C: (A) w/o Obj. Interactions	375	1.38	0.76
D: (A) w/o Sup.	301	1.13	0.84

Table 4.1: Ablation study results on Cityscapes.

4.3.1 Ablation Study

We conduct an ablation study on Cityscapes to evaluate the impact of the individual components of the model. We begin by testing the contribution of our GCN by removing the motion estimation module and directly use the object location tensor of the user-controlled object $\mathbf{B}_t$ in the appearance encoder. Table 4.1-B shows that removing the motion estimator leads to a drop in all three metrics. Without the GCN, the network cannot infer the positions of the objects in the scene and fails at moving the object. The quality of the video decreases as well (FVD 369 vs FVD 289).

Then, we test a version of the GCNs that does not model the interactions between objects. To do so, we remove all the edges between the nodes of the GCN, thus considering each object as independent. Tab. 4.1-C shows that,

Setting (N)	**Model**	**Cityscapes**			**KITTI 360**		
		FVD↓	NDE↓	Acc↑	FVD↓	NDE↓	Acc↑
	Sheng [103]	373	2.11	0.68	**443**	3.92	0.68
Oracle	Sheng*	498	2.12	0.58	507	3.66	0.66
(1)	S. Sheng*	493	1.78	0.57	527	3.79	0.33
	Ours	**288**	**1.01**	**0.84**	463	**1.83**	**0.75**
	Sheng [103]	373	1.53	0.66	**443**	3.98	0.62
Custom	Sheng*	498	1.61	0.57	506	3.27	0.60
(1)	S. Sheng*	493	1.41	0.59	527	3.34	0.30
	Ours	**303**	**0.66**	**0.88**	470	**2.06**	**0.81**
	Sheng [103]	373	1.48	0.73	**443**	2.93	0.48
Custom	Sheng*	498	1.47	0.67	506	3.19	0.49
(all)	S. Sheng*	493	1.38	0.60	527	2.71	0.24
	Ours	**321**	**0.96**	**0.86**	464	**1.58**	**0.72**

Table 4.2: Quantitative comparison in the *Oracle* and *Custom* setting. N is the number of user-controlled objects. $N = 1$ selects one object at random

while the object is correctly moved (NDE and Acc are similar to A), the video quality is considerably worse. In Appendix C, we qualitatively show that the network cannot move the other objects consistently.

Finally, we also test the network without flow supervision (*i.e.* Tab. 4.1-D). As expected, the performance decreases in NDE and FVD. Nevertheless, the quality of the image quality measured with FVD remains higher than when we do not model object interactions (*i.e.* Tab. 4.1-C).

4.3.2 Comparison with State-of-the-art

Quantitative comparison. We compare our method with the method of Sheng *et al.* [103], and its modifications, namely *Sheng** and *S. Sheng**. To the best of our knowledge, the method of Sheng *et al.* [103] model is the most similar work that generates videos in complex environments also leveraging

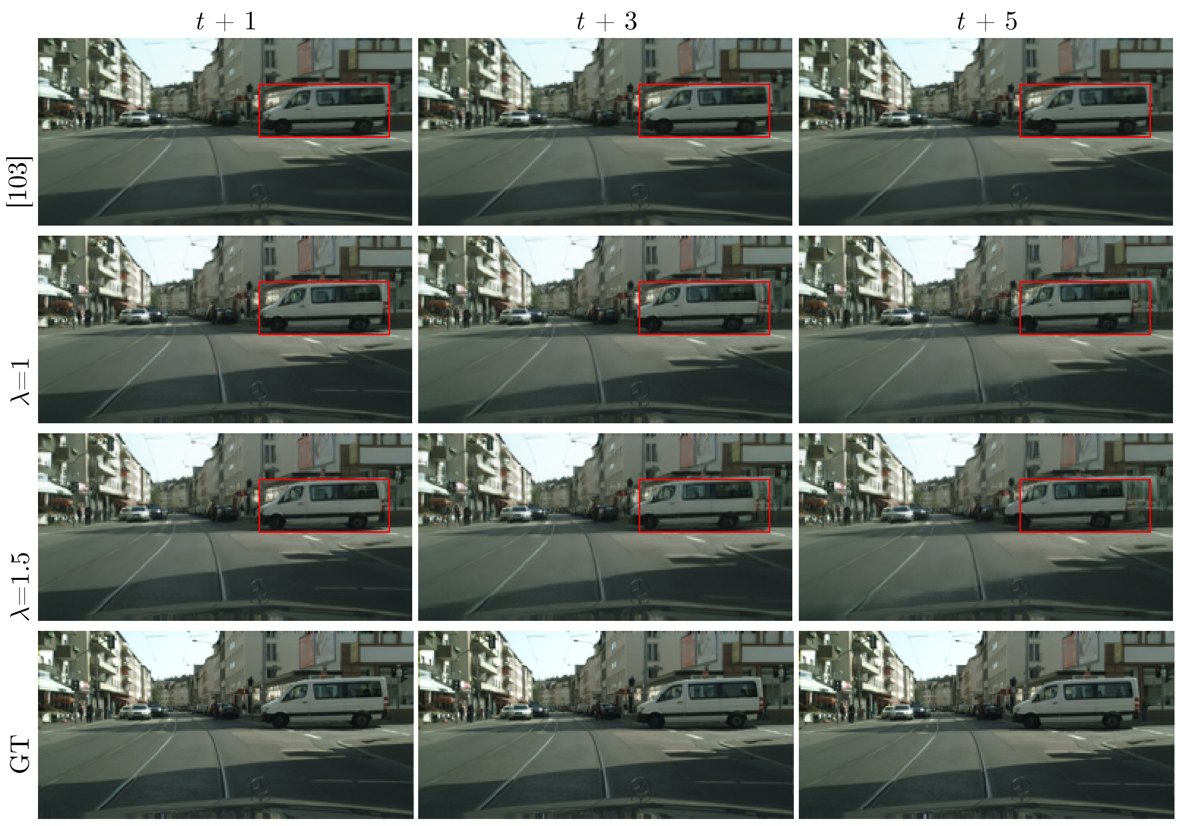

Figure 4.3: Qualitative comparison in the *Custom* setting on the Cityscapes dataset with ground truth reference. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details.

the semantic space of frames.

Tab. 4.2 shows the quantitative evaluation of all the models. We first compare our proposal in the *Oracle* setting, where the displacement $\boldsymbol{d}_n$ of one random object n is computed from ground truth frames. From NDE and Acc, we observe that our approach consistently outperforms state-of-the-art methods by enabling the user to move objects more precisely in both the datasets (see Tab. 4.2-Oracle results). In particular, NDE decreases from Sheng *et al.* [103] results by 47% and 53% in Cityscapes and KITTI 360, respectively. Regarding the video quality, which evaluates both the temporal consistency and image quality, we significantly improve the state-of-the-art performance in Cityscapes (FVD decreases by 22.79% from [103]), while in

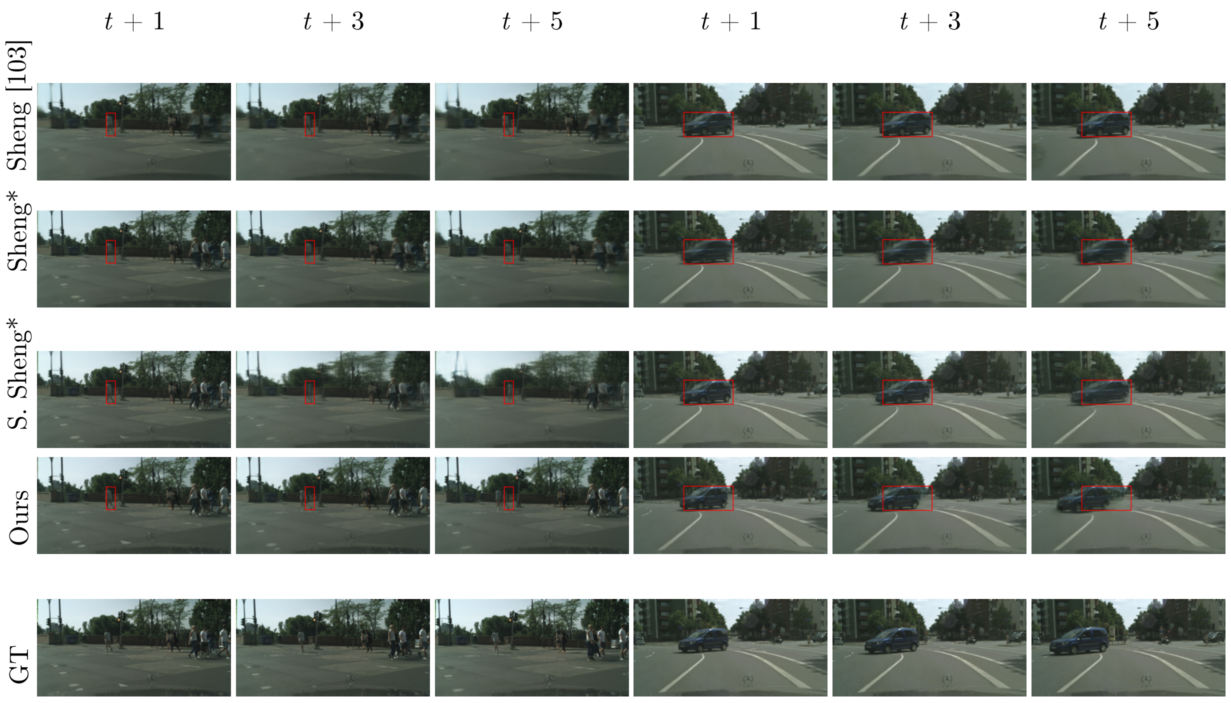

Figure 4.4: Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the Cityscapes dataset [14] with ground truth reference. On first three columns, we move the pedestrian near the semaphore to left. On the last three columns we move car crossing the street. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details.

KITTI 360 we are slightly worse than Sheng *et al.* [103]. We hypothesize this result is caused by the low frame rate of KITTI 360, which rewards Sheng *et al.* [103] that is dominated by modelling only the ego-motion, while ignoring other objects' movements. Through *Sheng** results, we note that adding the information to move the objects in the scene helps the baseline through NDE, but the video quality decreases significantly. Only through additional supervision (i.e. *S. Sheng**), FVD partially decreases. However, our model is far better at moving the objects in the scene, while having better video quality than *Sheng** and *S. Sheng**.

Tab. 4.2-*Custom* also shows the *Custom* experiment, where $\boldsymbol{d}_i$ is multiplied by $\lambda = 1.5$ from the ground truth displacement. Again, we observe that our proposal offers better control of the object's movements compared to state-of-the-art approaches and improves video quality. Our approach moves objects to positions that differ from the ground truth, showing that the *Motion Encoding* module correctly follows the user inputs, infers the missing objects and composes them in a temporally consistent manner.

We also perform experiments where we ask the models to move *all* objects (i.e. N objects) with the *Custom* setting. The last rows of Tab. 4.2 show that our proposal achieves the best results even at this “drastic" task.

Finally, we note that, our approach without supervised optical flow (Tab.4.1-D) outperforms the existing approaches compared in Tab. 4.2 both in terms of video quality and object control. This result confirms that the performance of our approach are not due to our use of supervision for optical flow but rather to our architecture.

Qualitative comparison. We now report the qualitative comparison for the tested models. Fig. 4.4 shows the results of two groups of experiments, where we feed the network with two different initial frames. In the first group of images, we want to move to the left the pedestrian that in the ground truth is in the position highlighted with the red bounding box. All three baselines fail to move the pedestrian. Sheng *et al.* [103] only moves the ego vehicle slightly to the right, leaving the pedestrian in the same position, while moving the entire scene. *Sheng** and *S. Sheng** moves the ego vehicle forward but fail at moving the pedestrian, which stays exactly in the same position in all the frames. C2M, instead, correctly and gradually moves to the left of the pedestrian, which goes out the red bounding box.

In the second group of images in the last three columns of Fig. 4.4, we aim to move a car that in the ground truth was in the position highlighted in red. Sheng *et al.* [103] can only move the ego-motion forward while the car remains in the same starting position. The other two baselines slightly move the car but not to the desired position specified by the user. However, our proposal significantly moves the car to the left, which goes partially out from the bounding box, while changing very little in the ego-motion of the video.

Finally, Fig. 4.3 shows a qualitative example of how our model can modify the van's position with different displacement. Moving it to the ground truth position ($\lambda = 1$) and to custom coordinates ($\lambda = 1.5$). As seen in the previous experiment, the baseline fails at moving the white van to the left. Instead, it stretches the back of the van. In contrast, with $\lambda = 1$ and $\lambda = 1.5$ the van goes from the bounding box with different horizontal shifts, depicting that our network can correctly change the position of the van to the user-specified positions.

4.4 Conclusion

In this chapter, we introduce *Click to Move*, a framework for video generation that allows the user to select key objects in the scene and control their motion by specifying their position in the last video frame. At test time, our approach receives the initial frame and the corresponding instance segmentation maps to generate a video that starts from the provided frame and respects the object motion constraints specified by the user. Objects in a scene are often not independent one from another. Thus, we introduce a novel GCN framework that employs specific message-passing rules to model object interaction while accounting for the user inputs. Experimentally, we demonstrate that our method outperforms state-of-the-art approaches and that the proposed GCN architecture allows better motion control. As future works, we plan to extend our approach to allow the generation of videos with variable length.

Chapter 5

Conclusion and Future Work

Deep generative models have shown promising results in video games' level generation [111, 117, 127], image-to-image translation [47, 70, 71], video-to-video translation [130], image inpainting [110, 143, 144] and video generation [85]. However, in conditional or constrained settings, unconditioned generative model fails to generate plausible objects because there is little to no control over the generated output.

In this thesis, we have explored three settings where unconstrained generative models may struggle.

In Chapter 2, we have presented Constrained Adversarial Networks (CANs) to solve the problem of structured objects generation. CANs are a generalization of vanilla GANs in which the constraints are embedded into the generator. Specifically, we made use of the Semantic Loss (SL) [139] to penalize the generator when it generates invalid structures during the training. Unlike other methods that use post-processing techniques, which affect the inference, the generation of valid structures only requires a forward pass of the network. Moreover, the compiled constraints used in the SL can be discarded after training, thus making possible to use CANs also on low-power devices.

Then, in Chapter 3 and Chapter 4 we shifted to multimedia manipulation.

Chapter 3 has explored the manipulation of images in challenging urban scenarios. Here, we have proposed an approach to unify the tasks of object insertion and image inpainting. In detail, we have introduced a novel method that alters a complex urban scene by removing a user specified portion of the image and coherently inserting a new object (e.g. car or pedestrian) in that scene. Our model is composed of two stages. In the first stage we give the user the possibility to either draw an object or to let the network decide its shape and position inside the missing area. The second part of the model consists of a one-stage architecture that leverages semantic segmentation to guide the inpainting of the missing part of the image. We test this model on two complex urban scenes datasets and we show that our proposed approach successfully address the problem of semantically-guided inpainting of complex urban scenes.

Finally, in Chapter 4 we have presented a novel framework that generates video sequences with object motion guidance. In this framework, the user can control the motion of the synthesized video through mouse clicks specifying simple object trajectories of the key objects in the scene. The movement of the objects that are not controlled by the user are modelled using a Graph Convolution Network (GCN). Then, the model combines an input frame, its corresponding segmentation map and the trajectories generated by the GCN, and generates a video sequence conditioned on the trajectories. Our experiments have shown that this novel method outperforms existing methods on two publicly available datasets, thus demonstrating the effectiveness of our GCN framework at modelling object interactions.

The works discussed in this thesis pave the way for various possible future directions. For example, in an ongoing work, we are investigating how we can improve the graph neural network proposed in Chapter 4. First of all, analysing various ways for improving the performance of our GCN, we have seen that objects that are detected as foreground but are static (e.g. parked cars) may have an impact on the performance of the predicted trajectories. Indeed, in datasets as Cityscapes, where parked cars or static pedestrians occupy a large portion of the scene, we have noticed that sometimes the objects move only by few pixels. To solve this issue, we are proposing to make use of deep learning techniques to detect static foreground objects and merge them into the background. This will allow our GCN to model the interaction of only dynamic objects in the scene. Then, we are working for extending our GCN in order to predict not only the object's position at time T, but also its scale and rotation. In particular, for each dynamic object, our GCN will predict its moving path, scale change, and shape in the future. The object appearance in the generated frames will be approximated by applying an affine transformation, predicted using spatial transformation network [48], on the object in the input frame.

Then, we plan to study how we can apply our graph neural network to video prediction [134, 25] or video interpolation [135]. In video prediction, given past frames, we aim at predicting future frames; while in video interpolation, the objective is predicting the intermediate frames between two frames. Although having multiple conditioning frames can lead to better ego-motion estimation and object's trajectories, these two scenarios are non trivial to solve if we want to put the human in the loop. Indeed, with respect to video generation, the user has less control on the possible trajectories that an object can take.

Following the great success of multimodal models in image synthesis [145] and video generation [41, 26], we are also planning to investigate possible solutions for the generation of new urban scenes and environments using other source of data. Thus, exploiting not only images and videos but also data on structural urban characteristics and social behaviors detected by other sources of data (e.g. location data or neighborhood characteristics). These models would have great impact in society since they will be able to understand the perception of cities.

Appendix A

Appendix to Chapter 2

A.1 Implementation details

A.1.1 Super Mario Bros Level Generation

The deep neural network for this experiment is based on the DCGANs used in [127]. Batch normalization and ReLU are applied between the layers of the generator g, while batch normalization and Leaky ReLU with a slope of 0.2 has been used for the discriminator d. In the last layer of the generator we apply a *softmax* activation function to obtain probabilities that are finally given in input to the Semantic Loss. On the other hand, the generation of samples is done through the application, always on g, of a stretched softmax function followed by an *argmax*, as in [127].

The networks have been trained using the WGAN guidelines [2]. Thus, the number of iterations on the discriminator has been set to 5 for each iteration on the generator. *RMSProp* has been used as optimizer, with a constant learning rate equal to 0.00005. The batch size used during the experiments has been set to 32. Layers have been initialized using *normal* initializer for both the generator and the discriminator. Moreover, weight clipping is applied on the weights of d with c equal to 0.01. Finally, the size of

the latent vector has been set to 32 and sampled from a normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$
Table A.1 shows the network architecture of g and d. In experiments where the SL is enabled only after some epoch e, in the first epochs before e only GANs runs are executed. After that threshold, CAN runs are created as a fork of GAN runs.

Part	Input Shape → Output Shape	Layer Type	Kernel	Stride
g	$(32) \rightarrow (1, 1, 32)$	Reshape.	-	-
	$(1, 1, 32) \rightarrow (\frac{h}{8}, \frac{w}{8}, 16)$	Deconv.	4×4	1×1
	$(\frac{h}{8}, \frac{w}{8}, 16) \rightarrow (\frac{h}{4}, \frac{w}{4}, 8)$	Deconv.	4×4	2×2
	$(\frac{h}{4}, \frac{w}{4}, 8) \rightarrow (\frac{h}{2}, \frac{w}{2}, 4)$	Deconv.	4×4	2×2
	$(\frac{h}{2}, \frac{w}{2}, 4) \rightarrow (h, w, 13)$	Deconv.	4×4	2×2
d	$(h, w, 13) \rightarrow (\frac{h}{2}, \frac{w}{2}, 64)$	Conv.	4×4	2×2
	$(\frac{h}{2}, \frac{w}{2}, 64) \rightarrow (\frac{h}{4}, \frac{w}{4}, 128)$	Conv.	4×4	2×2
	$(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{8}, \frac{w}{8}, 256)$	Conv.	4×4	2×2
	$(\frac{h}{8}, \frac{w}{8}, 256) \rightarrow (1, 1, 1)$	Conv.	4×4	1×1

Table A.1: Super Mario Bros Level Generation network architecture.

Details about the *pipes* constraint

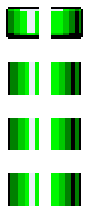

Figure A.1: A decomposed pipe

As reported in Section 2.4, experiment with the constraint on pipes has been run for 12000 epochs. Figure A.1 shows the various parts composing a pipe and their disposition. Suppose to call $\mathbf{a}$ the matrix boolean variables corresponding to the output of the generator $\mathbf{x}$ with shape $2 \times 2 \times$. Remember that we apply the constraint separately to windows of size 2×2, with each pixel having 5 channels. The four channels represent the probabilities of the tiles: [*top-left*, *top-right*, *body-left*, *body-right*, *others*]. In particular, in the last channel we collapse all the probabilities of the tiles that do not belong to pipes (air, monsters, walls, ...). Then, given the $2 \times 2 \times 5$ boolean vector, the list of the clauses composing the final constraint can be written as:

$(\mathbf{a}_{0,0,0} \iff \mathbf{a}_{0,1,1})$	top-left tile requires top-right tile on the right and vice-versa
$(\mathbf{a}_{0,0,2} \iff \mathbf{a}_{0,1,3})$	body-left tile requires body-right tile on the right and vice-versa
$(\mathbf{a}_{0,0,0} \rightarrow \mathbf{a}_{1,0,2})$	top-left tile requires body-left tile below
$(\mathbf{a}_{0,1,1} \rightarrow \mathbf{a}_{1,1,3})$	top-right tile requires body-right tile below
$(\mathbf{a}_{1,0,2} \rightarrow (\mathbf{a}_{0,0,2} \vee \mathbf{a}_{0,0,0}))$	body-left tile requires body-left of top-left above
$(\mathbf{a}_{1,1,3} \rightarrow (\mathbf{a}_{0,1,3} \vee \mathbf{a}_{0,1,1}))$	body-right tile requires body-right of top-right above
$\bigwedge_{i=0}^{1} \bigwedge_{j=0}^{1} \mathrm{OHE}(\mathbf{a}_{i,j})$	One hot encoding over all the 4 positions

Notice that first two indexes describe the position, e.g. $0, 0$ means the upper left corner of the 2×2 window, and the third index defines the tile type.

Details about the *reachability* constraint

The feedforward neural network ϕ is implemented using a CNN with two final dense layers, which architecture is described in Table A.2.

Performances of the approximation network ϕ include an accuracy and an F1-score higher than 94%. Picture A.2 shows examples of how reachability maps have been approximated with ϕ. The first column contains levels generated by the GAN. The binary maps have been obtained by summing the probabilities of all the solid tiles (ground, pipes, ...), given in white. The second column contains the reachability maps, computed by the $A*$ agent and averaged over 100 different runs. Finally, the third column shows the

Input Shape → Output Shape	**Layer Type**	**Kernel**	**Stride**
$(h, w, 13) \to (h, w, 8)$	Conv.	3×3	1×1
$(h, w, 8) \to (h, w, 16)$	Conv.	5×5	1×1
$(h, w, 16) \to (h, w, 24)$	Conv.	7×7	1×1
$(h, w, 24) \to (h, w, 32)$	Conv.	9×9	1×1
$(h, w, 32) \to (h, w, 64)$	Conv.	11×11	1×1
$(h, w, 64) \to (h, w, 96)$	Conv.	13×13	1×1
$(h, w, 96) \to (h, w, 128)$	Conv.	15×15	1×1
$(h, w, 128) \to (h, w, 192)$	Conv.	17×17	1×1
$(h, w, 192) \to (h, w, 32)$	Dense	-	-
$(h, w, 32) \to (h, w, 2)$	Dense	-	-

Table A.2: Reachability Network architecture.

reachability maps approximated by the ϕ neural network. Notice how well does ϕ work: the second and third columns are almost indistinguishable. P_R and $\hat{P_R}$ are such that $A* : P_g \to P_R$ and $\phi : P_g \to \hat{P_R}$

A.1.2 Molecule Generation

The MolGAN architecture is composed of three networks, the generator G, the discriminator D and the reward network R. D and R share the same architecture, but are trained with different objectives. R is trained to predict the product of the QED, SA, logP metrics (in $[0, 1]$). G is optimized to produce samples that maximize the output of R and are convincing to D, on top of that, the conditional semantic loss is also applied.
While R is trained in parallel with D and G, the loss of G with respect to the output of R is activated only after 150 epochs, at which point the adversarial loss stops being used. The semantic loss is applied to G from the start to the end of the training. The weight of the semantic loss is equal to 0.9, whereas the adversarial loss of G has a weight of 0.1.
Improved WGAN is used as the adversarial loss between G and D ($n_{critic} = 5$), whereas R is trained to estimate the desired target by mean squared error,

and G is trained with deep deterministic policy gradient w.r.t. the output of R, which is seen as a reward to maximize.

Training proceeds until the uniqueness of the batch falling below 0.2. During training, the learning rate is set at a constant value of 0.001, the batch size is 32 and there is no dropout. Adam with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ is the optimizer of choice. Batch discrimination is used.

Results are obtained by evaluating a batch of 5000 generated samples.

The input noise z has dimension 32 if the semantic loss is not applied, 36 if it is applied; with the first 32 dimensions sampled from a standard normal distribution and the last four from a uniform distribution in $[0, 1]$.

The maximum number of nodes for each molecule is 9, with 5 possible atom types and 5 bond types. Each molecule is represented by two matrices, one mapping each node to a label, and one adjacency matrix informing about the presence or lack of edges between nodes, and their type.

The input noise is received by G and processed by three fully connected layers of $128, 256, 512$ units each, while tanh acts as the activation function; a linear projection followed by a softmax is then applied to the output of the last layer to have it matching the size of the adjacency matrices.

D and R share the same architecture (no parameters are shared), based on two relational graph convolutions [18] of $64, 32$ hidden units, followed by an aggregation as in [18] to obtain a graph level representation of 128 features. Two fully connected layers of $128, 1$ units then reduce the graph embedding to a single output value, with tanh as the activation for the hidden layer and with sigmoid being applied on the output in the case of R. The MolGAN architecture is trained on the QM9 dataset [90, 95] composed of 5000 training examples.

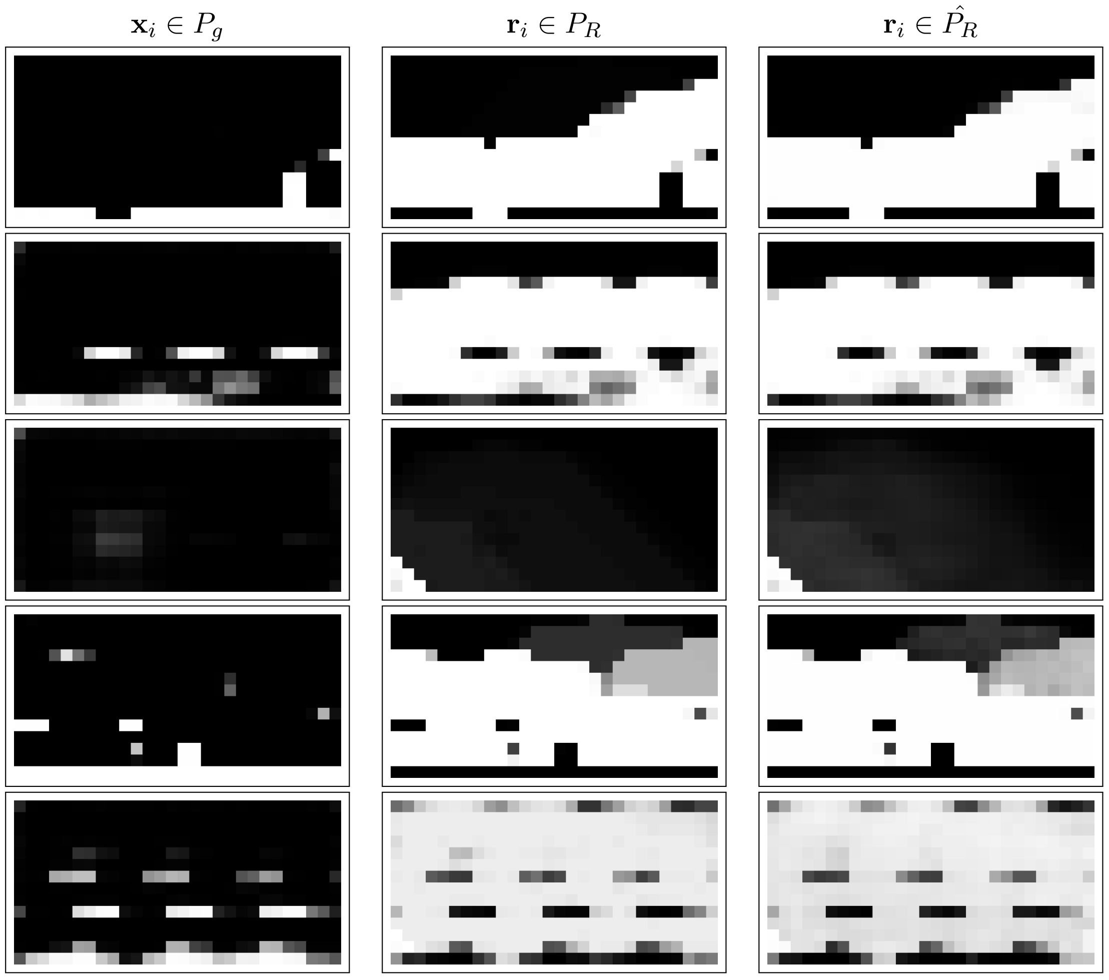

Figure A.2: Given some generated levels (first column), this picture compares real reachability maps computed by the A^* agent (second column) and approximated by $\hat{P_R}$ (third column).

Appendix B

Appendix to Chapter 3

B.1 Implementation Details

The encoder of our image completion architecture follows the two-stream convolutional encoder of [40]. Specifically, both encoders downsample the input four times, using Instance normalization [121] and ELU [13] as activation function. The encoder is followed by nine residual blocks. With respect to the original proposed residual block, we have decided to use dilated convolution with increasing dilation factor as proposed in [142], Instance normalization [121], ELU activation [13]. Moreover we have decided to remove the activation function at the end of each residual block after an evaluation phase. Out proposed model has been implemented using PyTorch. The batch has been set to four due to GPU limitation. The model has been trained for 600 epochs for the Cityscapes dataset and for 200 epochs for the Indian Driving Dataset using Adam [55] as optimizer with a learning rate of 0.0002. For our experiments we choose $\lambda_{recon} = \lambda_{perc} = \lambda_{FM} = \lambda_{cross} = 10$, $\lambda_{style} = 250$, $\lambda_{VAE} = 5$ and $\lambda_{inst\_recon} = 20$. Spectral normalization (SN) [79] was used in in the discriminator. Although recent works [146] have shown that SN can be used also for the generator, it further increases the training time with only minor improvements. For this reason, we have decided to not use it.

The details of the architecture are shown in Table B.1 and Table B.2. We release the source code of our model and the SPG-Net implementation at `https://github.com/PierfrancescoArdino/SGINet`.

Part	Input → Output Shape	Layer Information
E_s	(4104,) → (8128,)	FC-(8128), LeakyReLU
	(, 8128) → (32, 32, 8)	RESHAPE
	(32, 32, 8) → (32, 32, 32)	CONV-(N32, K3x3, S1, P1), IN, LeakyReLU
	(32, 32, 32) → (16, 16, 64)	CONV-(N64, K4x4, S2, P1), IN, LeakyReLU
	(16, 16, 64) → (8, 8, 128)	CONV-(N128, K4x4, S2, P1), IN, LeakyReLU
	(8, 8, 128) → (4, 4, 256)	CONV-(N256, K4x4, S2, P1), IN, LeakyReLU
	(4,4, 256) → (Z,)	CONV-(Z, K4x4, S1, P0)
	(4,4 256) → (Z,)	CONV-(Z, K4x4, S1, P0)
G_s	$(Z + N_C + \theta) \rightarrow (\frac{h}{16}, \frac{h}{16}, 256)$	DECONV-(N256, K4x4, S1, P0), IN, ReLU
	$(\frac{h}{16}, \frac{h}{16}, 256) \rightarrow (\frac{h}{8}, \frac{h}{8}, 128)$	DECONV-(N128, K4x4, S2, P1), IN, ReLU
	$(\frac{h}{8}, \frac{h}{8}, 128) \rightarrow (\frac{h}{4}, \frac{h}{4}, 64)$	DECONV-(N64, K4x4, S2, P1), IN, ReLU
	$(\frac{h}{4}, \frac{h}{4}, 64) \rightarrow (\frac{h}{2}, \frac{h}{2}, 32)$	DECONV-(N32, K4x4, S2, P1), IN, ReLU
	$(\frac{h}{2}, \frac{w}{2}, 32) \rightarrow (h, w, 1)$	CONV-(N32, K4x4, S2, P1), SIGMOID
D_s	$(h,w,1) \rightarrow (\frac{h}{2},\frac{w}{2},64)$	CONV-(N64, K4x4, S2, P1), LeakyReLU
	$(\frac{h}{2},\frac{w}{2},64) \rightarrow (\frac{h}{4},\frac{w}{4},128)$	CONV-(N128, K4x4, S2, P1), IN, LeakyReLU
	$(\frac{h}{4},\frac{w}{4},128) \rightarrow (\frac{h}{8},\frac{w}{8},256)$	CONV-(N256, K4x4, S2, P1), IN, LeakyReLU
	$(\frac{h}{8},\frac{w}{8},256) \rightarrow (\frac{h}{16},\frac{w}{16},512)$	CONV-(N512, K4x4, S2, P1), IN, LeakyReLU
	$(\frac{h}{16},\frac{w}{16},256) \rightarrow (1,1,1)$	CONV-(N1, K4x4, S1, P0)

Table B.1: Network architecture. We use the following notation: Z: the dimension of attribute vetor, N_C: the number of the class instance, θ: the number of location parameters of the affine transformation, K: kernel size, S: stride size, P: padding size, CONV: a convolutional layer, DECONV: a deconvolutional layer, FC: fully connected layer, IN: Instance Normalization

Part	**Input → Output Shape**	**Layer Information**
	$(h, w, 3 + \boldsymbol{m}) \rightarrow (h, w, 32)$	CONV-(N32, K5x5, S1, P2), IN, ELU
	$(h, w, 32) \rightarrow (\frac{h}{2}, \frac{w}{2}, 64)$	CONV-(N64, K5x5, S1, P2), IN, ELU
E_{im}	$(\frac{h}{2}, \frac{w}{2}, 64) \rightarrow (\frac{h}{4}, \frac{w}{4}, 128)$	CONV-(N128, K4x4, S2, P1), IN, ELU
	$(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{8}, \frac{w}{8}, 256)$	CONV-(N256, K4x4, S2, P1), IN, ELU
	$(\frac{h}{8}, \frac{w}{8}, 256) \rightarrow (\frac{h}{16}, \frac{w}{16}, 512)$	CONV-(N512, K4x4, S2, P1), IN, ELU
	$(h, w, \mathcal{C} + \boldsymbol{m} + \hat{\mathbf{m}}_s) \rightarrow (h, w, 32)$	CONV-(N32, K5x5, S1, P2), IN, ELU
	$(h, w, 32) \rightarrow (\frac{h}{2}, \frac{w}{2}, 64)$	CONV-(N64, K5x5, S1, P2), IN, ELU
E_{se}	$(\frac{h}{2}, \frac{w}{2}, 64) \rightarrow (\frac{h}{4}, \frac{w}{4}, 128)$	CONV-(N128, K4x4, S2, P1), IN, ELU
	$(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{8}, \frac{w}{8}, 256)$	CONV-(N256, K4x4, S2, P1), IN, ELU
	$(\frac{h}{8}, \frac{w}{8}, 256) \rightarrow (\frac{h}{16}, \frac{w}{16}, 512)$	CONV-(N512, K4x4, S2, P1), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 512 + 512) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	CONCAT(E_{im}, E_{se})
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D2), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D2), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D2), IN, ELU
E	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D4), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D4), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D4), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D8), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D8), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{16}, \frac{w}{16}, 1024)$	Residual Block: CONV-(N1024, K3x3, S1, P1, D8), IN, ELU
	$(\frac{h}{16}, \frac{w}{16}, 1024) \rightarrow (\frac{h}{8}, \frac{h}{8}, 512)$	Decoder Block
	$(\frac{h}{8}, \frac{w}{8}, 512) \rightarrow (\frac{h}{4}, \frac{h}{4}, 256)$	Decoder Block
G	$(\frac{w}{16}, \frac{w}{16}, 256) \rightarrow (\frac{h}{2}, \frac{w}{2}, 128)$	Decoder Block
	$(\frac{h}{2}, \frac{w}{2}, 128) \rightarrow (h, w, 64)$	Decoder Block
	$(h, w, 64) \rightarrow (h, w, 32)$	SPADEResBlock
	$(h, w, 32) \rightarrow (h, w, 3)$	CONV-(N32, K7x7, S1, P3, D8), Tanh
	$(h, w, 3 + \mathcal{C}) \rightarrow (\frac{h}{2}, \frac{w}{2}, 64)$	CONV-(N64, K4x4, S2, P1), Leaky ReLU, SN
D_g	$(\frac{h}{2}, \frac{w}{2}, 64) \rightarrow (\frac{h}{4}, \frac{w}{4}, 128)$	CONV-(N128, K4x4, S2, P1), IN, Leaky ReLU, SN
	$(\frac{h}{4}, \frac{w}{4}, 128) \rightarrow (\frac{h}{8}, \frac{w}{8}, 256)$	CONV-(N256, K4x4, S2, P1), IN, Leaky ReLU
	$(\frac{h}{8}, \frac{w}{8}, 256) \rightarrow (\frac{h}{8}, \frac{w}{8}, 512)$	CONV-(N512, K4x4, S1, P1), IN, Leaky ReLU
	$(\frac{h}{8}, \frac{w}{8}, 512) \rightarrow (\frac{h}{8}, \frac{w}{8}, 1)$	CONV-(N1, K4x4, S1, P1)

Table B.2: Network architecture. We use the following notation: $\boldsymbol{m}$: the dimension of the mask, $\mathcal{C}$: the semantic classes, $\hat{\mathbf{m}}_s$: the dimension of the instance, K: kernel size, S: stride size, P: padding size, D: dilation factor, CONV: a convolutional layer, IN: Instance Normalization, SN: Spectral Normalization

B.2 Dataset details

Cityscapes. The dataset contains 5000 street level images divided between training, validation and testing sets. We used the 2975 images from the training set during the training and we tested our model on the 500 images from the validation set. The resolution of the images is 2048×1024. We resize each image to be 512×256 in order to maintain the scale and then we apply random cropping of size 256×256. Cityscapes segmentation maps have 35 categories. We aggregated them in 17 categories inspired by the literature [14].

Indian Driving. It consists of 20000 images collecting driving sequences on Indian roads. As for the Cityscapes, we used the validation set for the evaluation phase. We removed the 720p images and resized the remaining (1080p) images to 512×288. Thus, we obtain a dataset of 11564 training images and 1538 evaluation images. Among these evaluation images, we randomly select 500 for the evaluation. Finally, we apply random cropping of size 256×256. To be comparable with Cityscapes, we aggregated the 40 segmentation maps categories into 21 groups.

B.3 Use cases

We test the network also on different uses cases listed in Figure 3.1, namely *precise removal* and *mask insertion.* The former aims to remove the exact shape of the object by reconstructing background pixels, while the latter inserts a mask provided by the user as input. We observe that our solution outperforms the state of the art both quantitatively (see Table B.3) and qualitatively (see Figure B.2 and Figure B.1).

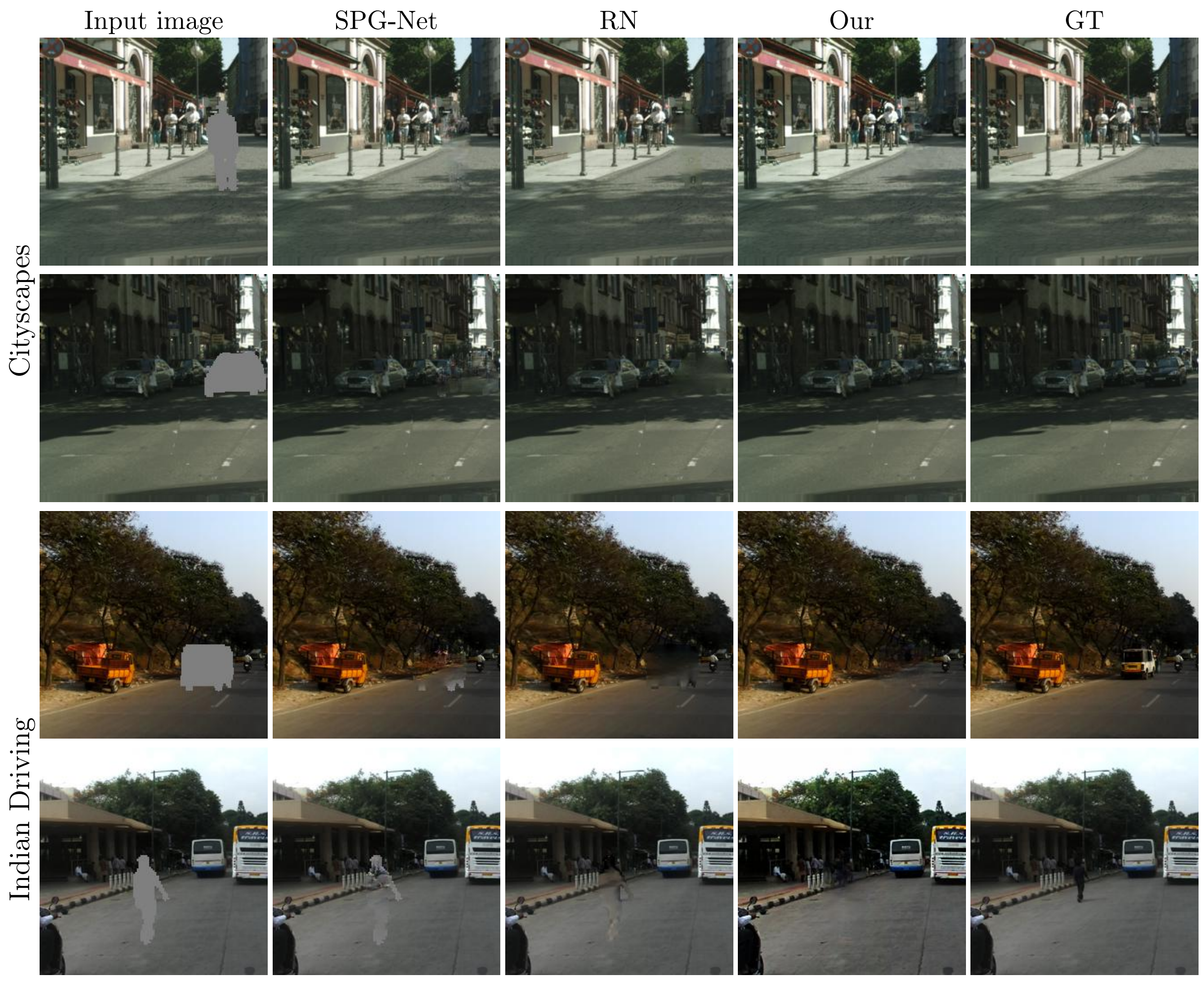

Figure B.1: Qualitative evaluation on the task of object precise removal. The first two rows show results on the Cityscape dataset while the last two rows show the Indian Driving results. We show two types of object removal: cars and pedestrians.

B.4 Additional results

We show additional result for each dataset and for each experimental settings. In particular, Figure B.3 and Figure B.4 show additional results for both the the Cityscapes and the Indian Driving datasets for the *reconstruct* experimental settings. While, Figure B.5 and Figure B.6 for the *insert & reconstruct* experimental settings.

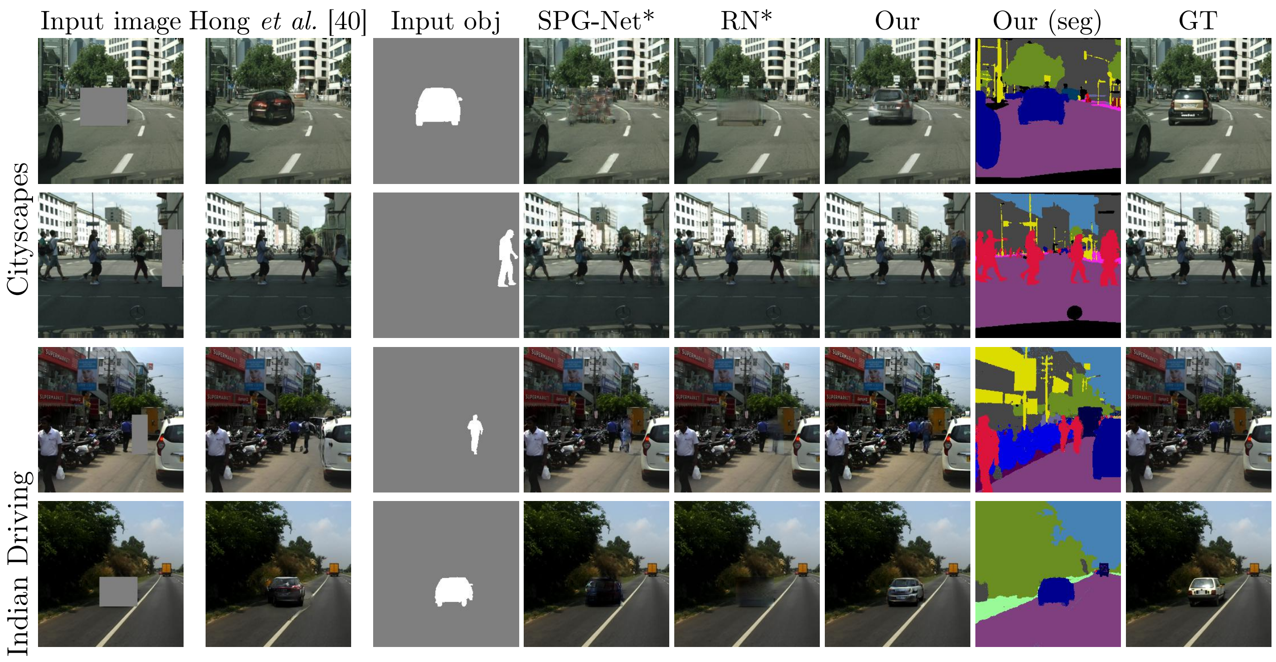

Figure B.2: Qualitative evaluation on the task of "mask insertion" object reconstruction. The first two rows show results on the Cityscape dataset while the last two rows show the Indian Driving results. We show two types of object insertion: cars and pedestrians.

Table B.3: Quantitative results for our model and the baselines for the use cases of precise removal and maks insertion. We evaluate all the networks in two experimental settings and through image quality (PSNR and FID).

	Model	**Cityscapes**		**Indian Driving**	
		PSNR↑	FID↓	PSNR↑	FID↓
Restore	Hong *et al.* [40]				
	SPG-Net [110]	29.35	21.56	25.06	42.68
	RN [144]	30.34	23.41	27.04	50.20
	Our proposal	**31.06**	**16.34**	**27.20**	**40.15**
Restore	Hong *et al.* [40]	31.03	13.87	28.23	19.20
	SPG-Net*	31.36	17.96	27.44	26.96
	RN*	31.70	18.67	28.32	39.91
	Our proposal	**32.39**	**10.96**	**28.85**	**19.02**

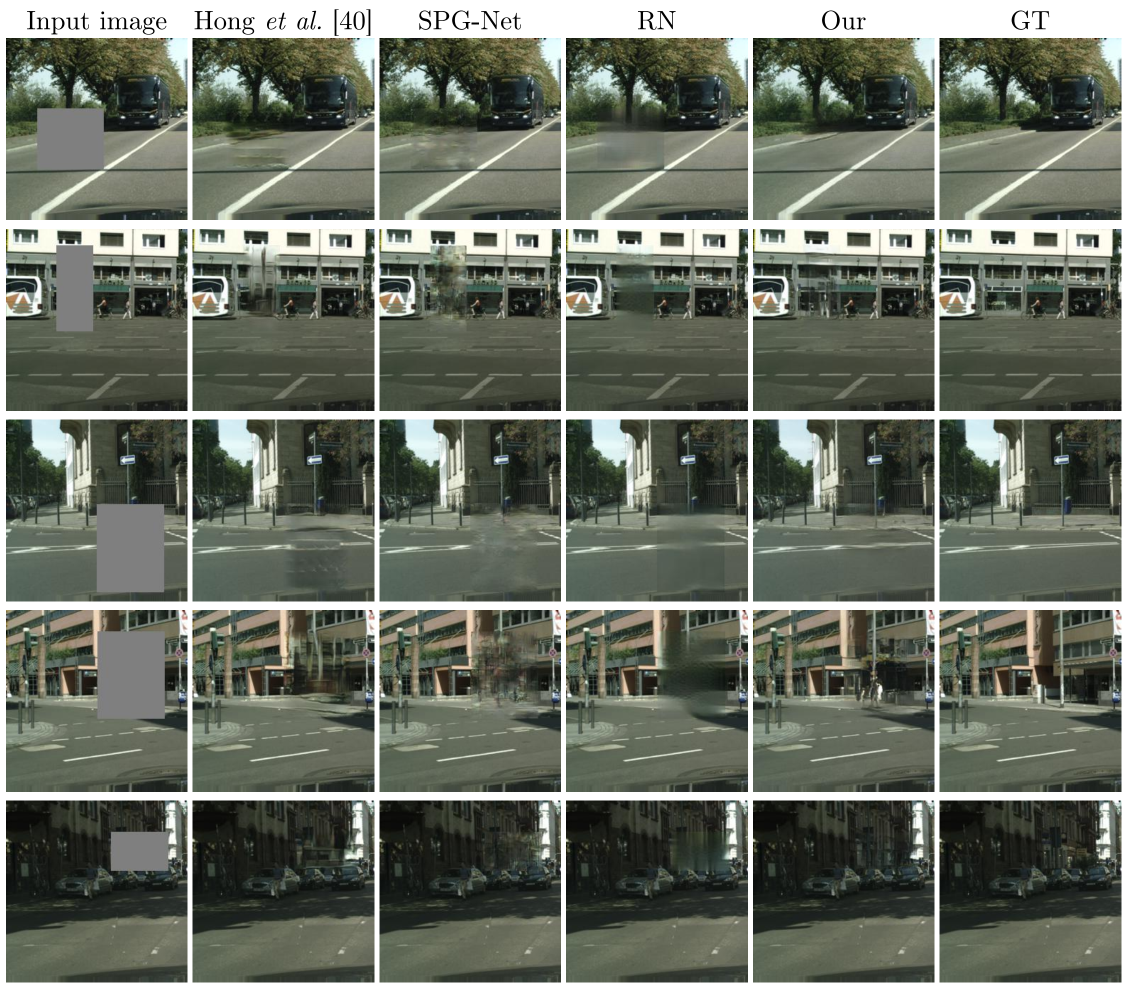

Figure B.3: Additional results on Cityscapes reconstruct experimental setting.

B.5 Edges contours

In complex scenes, conventional inpainting methods might generate unsatisfactory results. For example, edge generators might underperform when the missing areas are either big or when they contain multiple objects and semantics. Thus, we show EdgeConnect [80] results, which is a popular inpaiting method that exploits edges to inpaint missing regions. Figure B.7 shows the results of the edge inpainting in Cityscapes. As can be seen, when the portion of image to inpaint is large, the network is not capable of generate plausible edges. Indeed, in the first row it fails to generate the border of the

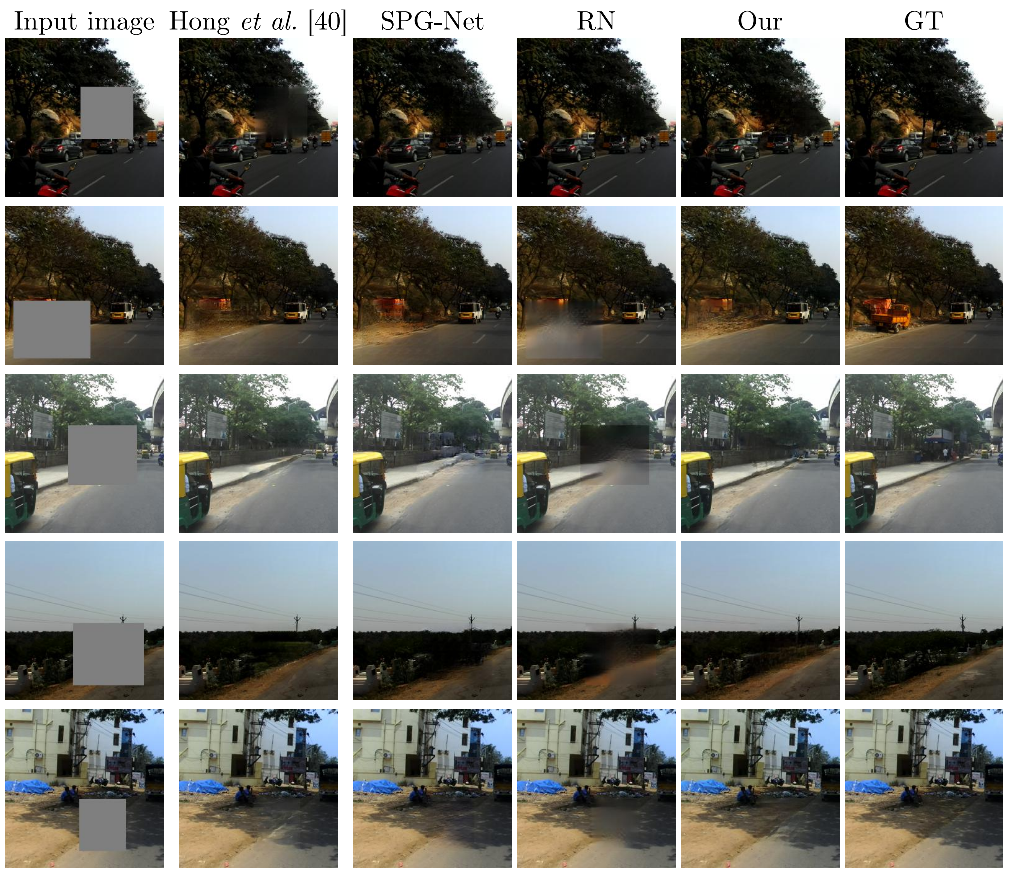

Figure B.4: Additional results on Indian Driving Dataset reconstruct experimental setting.

building. While in the second row it generates a huge quantity of lines that do not fit the context of the object. Generated edges in complex scenes are often not reliable.

B.6 Additional ablations

As additional tests, we also ablate the different components of our decoder block. In our decoder, we removed the learned convolution for the residual skip connection, and replaced the Batch Normalization [46] with Instance Normalization [120]. Table B.4 also shows that our modifications to the original SPADE [86] are effective. Moreover, they allow using this block when the batch size is small (or even one).

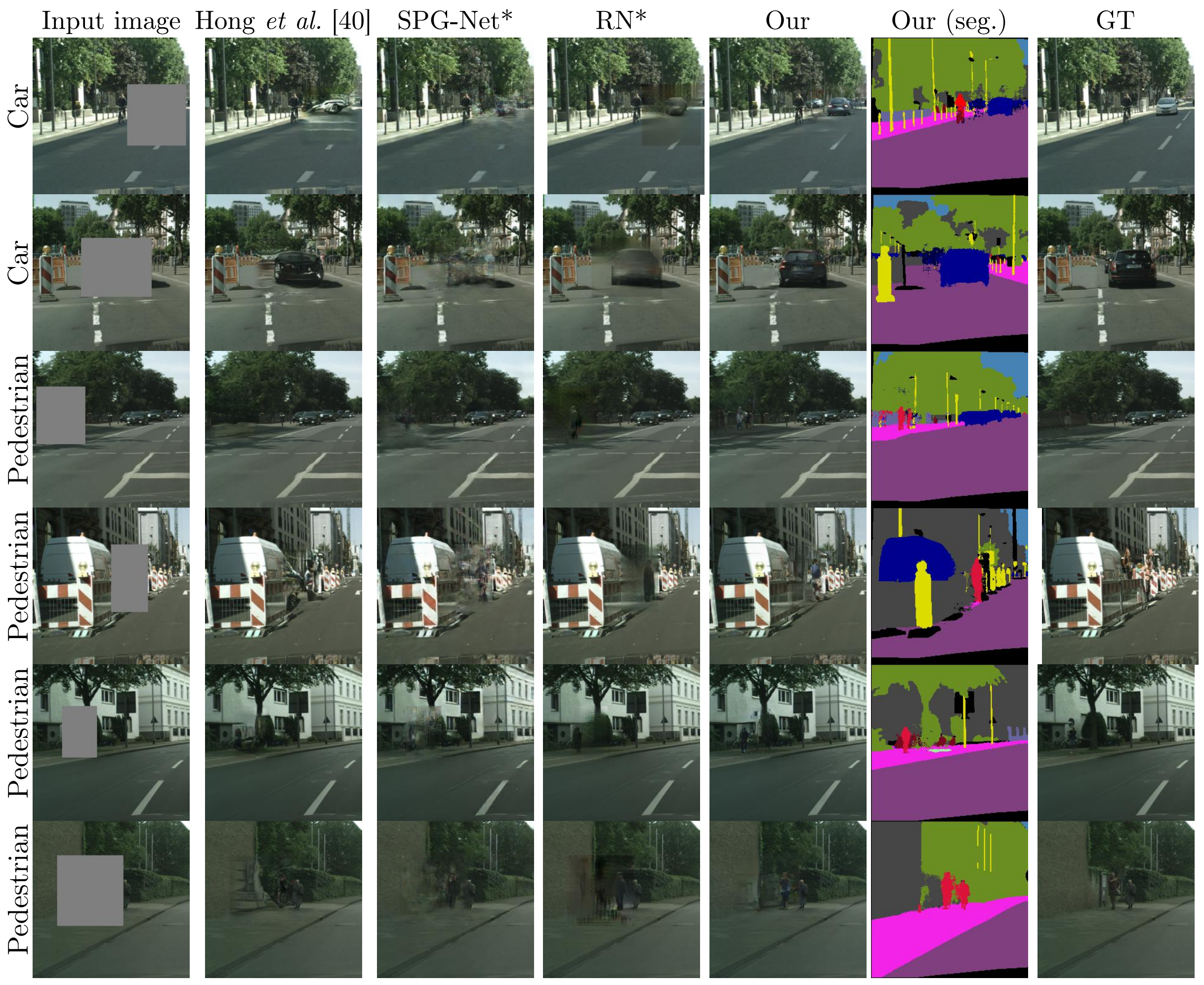

Figure B.5: Additional results on Cityscapes Dataset insert & reconstruct experimental setting.

In Figure B.8 we instead show the qualitative evaluation on the ablation study with and without $\mathcal{L}_{\text{style}}$. The figure shows that removing $\mathcal{L}_{\text{style}}$ the network generate "checkerboard" artifacts.

Table B.4: Ablation study of the components of the SPADE block.

Model	**PSNR↑**	**FID↓**
Our proposal (A)	**32.96**	**5.05**
(A) w/o SPADE	32.57	5.56
(A) w Sync Batch Normalization	32.05	5.59
(A) w learned skip connection	32.76	5.49

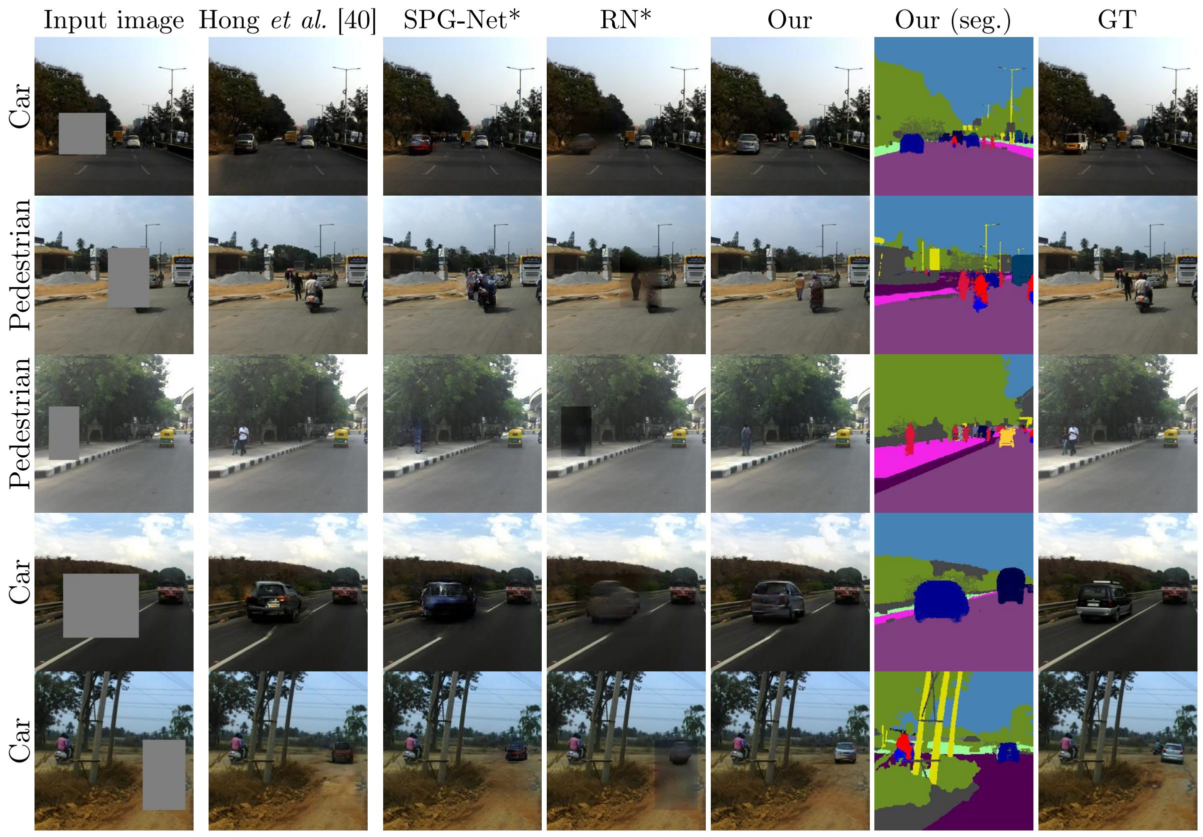

Figure B.6: Additional results on Indian Driving Dataset insert & reconstruct experimental setting.

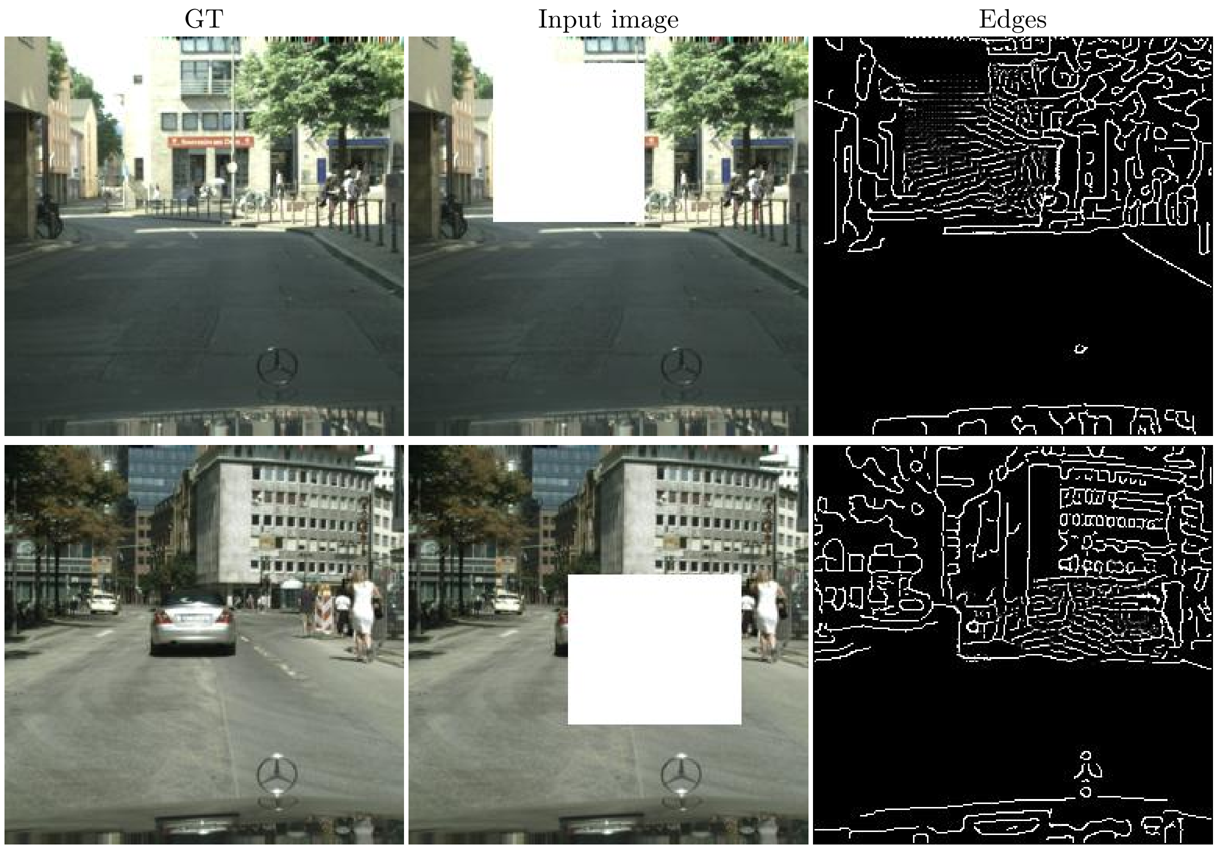

Figure B.7: Results of [80] on the Cityscapes dataset.

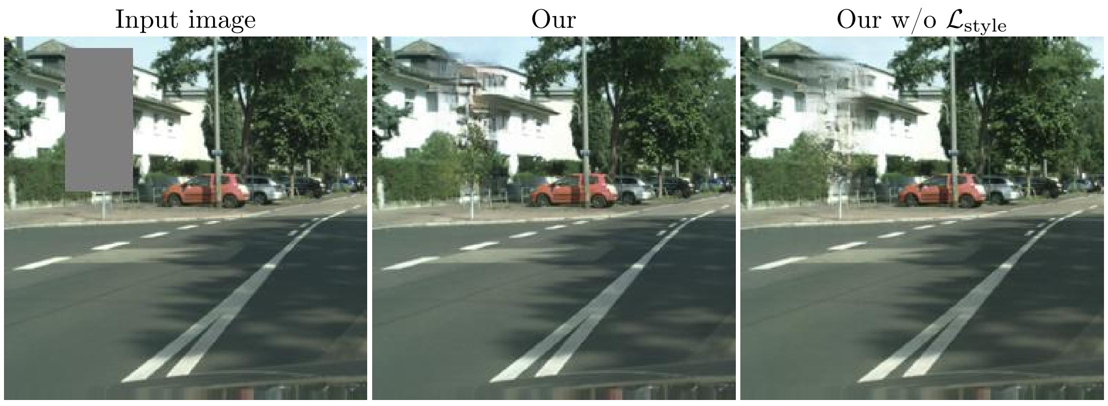

Figure B.8: Qualitative evaluation on the ablation study with and without $\mathcal{L}_{\text{style}}$. Removing $\mathcal{L}_{\text{style}}$ the network generate "checkerboard" artifacts. Zoom in for better details.

Appendix C

Appendix to Chapter 4

We here provide additional information regarding network architectures (Appendix C.1) and implementation details (Appendix C.2). Then, we provide additional high-resolution qualitative results in Appendix C.3 on both Cityscapes and KITTI 360.

C.1 Network architectures

We provide the details of the trajectory encoder E_s and flow predictor D in Table C.1. The flow decoder is composed of two heads (referred to as $Flow_D$ and Occ_D) and shared layers (referred to as $Feat_D$). For more details regarding the architecture of the other networks, please refer to the code attached with this supplementary material.

C.1.1 Convergence issue

During preliminary experiments, we observed that using eq.(1) update rule (from the manuscript) ended up with all the nodes having the exact same features. For this reason, we added a residual update that helped objects converging to better features. Indeed, this residual update can be seen as

Part	Input → Output Shape	Layer Information
E_s	(BS,5, 64,128, 1) → (BS,5, 32,64,32)	3DCONV-(N32, K{3,4,4}, S{1,2,2}, P{1,1,1}), BN, LeakyReLU
	(BS,5, 32,64,32) → (BS,5, 16,32,64)	3DCONV-(N64, K{3,4,4}, S{1,2,2}, P{1,1,1}), BN, LeakyReLU
	(BS,5, 16,32,64) → (BS,5, 8,16,128)	3DCONV-(N128, K{3,4,4}, S{1,2,2}, P{1,1,1}), BN, LeakyReLU
$Feat_D$	(BS * 5, 2, 4, 272) → (BS * 5, 2, 4, 512)	CONV-(N512, K3, S1, P1), BN, LeakyReLU
	(BS * 5, 2, 4, 512) → (BS * 5, 4, 8, 256)	UP, CONV-(N256, K3, S2, P1), BN, LeakyReLU
	(BS * 5, 4, 8, 512) → (BS * 5, 8, 16, 128)	SKIP, UP, CONV-(N128, K3, S1, P1), BN, LeakyReLU
	(BS * 5, 4, 8, 512) → (BS, 5, 8, 16, 128)	RESHAPE
	(BS, 5, 8, 16, 256) → (BS, 5, 8, 16, 128)	SKIP, 3DCONV-(N128, K3, S1, P1), BN, LeakyReLU
	(BS, 5, 8, 16, 128) → (BS * 5, 8, 16, 128)	RESHAPE
	(BS * 5, 8, 16, 256) → (BS * 5, 16, 32, 64)	SKIP, UP, CONV-(N64, K3, S1, P1), BN, LeakyReLU
	(BS * 5, 16, 32, 64) → (BS, 5, 16, 32, 64)	RESHAPE
	(BS, 5, 16, 32, 128) → (BS, 5, 16, 32, 64)	SKIP, 3DCONV-(N64, K3, S1, P1), BN, LeakyReLU
	(BS, 5, 16, 32, 64) → (BS * 5, 16, 32, 64)	RESHAPE
	(BS * 5, 16, 32, 128) → (BS * 5, 32, 64, 32)	SKIP, UP, CONV-(N32, K3, S1, P1), BN, LeakyReLU
	(BS * 5, 32, 64, 32) → (BS, 5, 32, 64, 32)	RESHAPE
	(BS, 5, 32, 64, 64) → (BS, 5, 32, 64, 32)	SKIP, 3DCONV-(N32, K3, S1, P1), BN, LeakyReLU
$Flow_D$	(BS * 5, 32, 64, 64) → (BS * 5, 64, 128, 32)	SKIP, UP, CONV-(N32, K3, S1, P1), BN, LeakyReLU
	(BS * 5, 64, 128, 32) → (BS * 5, 64, 128, 2)	CONV-(N2, K5, S1, P2), IN, Tanh
Occ_D	(BS * 5, 32, 64, 64) → (BS * 5, 64, 128, 32)	SKIP, UP, CONV-(N32, K3, S1, P1), BN, LeakyReLU
	(BS * 5, 64, 128, 32) → (BS * 5, 64, 128, 2)	CONV-(N1, K5, S1, P2), Sigmoid

Table C.1: Network architecture. We use the following notation: Z: the dimension of motion vector, K: kernel size, S: stride size, P: padding size, CONV: a convolutional layer, UP: upsample, 3D-CONV: 3D convolutional layer, BN: Batch Normalization, SKIP: skip connection

skip connections, similar to those of resnet architectures, that allow gradient information to pass through the GCN updates and mitigate vanishing gradient problems.

C.2 Implementation.

Our architecture is implemented with Pytorch 1.7.0, while the graph neural network has been implemented using Pytorch Geometric. We use the ADAM optimizer [55] with a learning rate of 2e-4 for the Generation module and 1e-4 for the Motion estimation. The modules are trained upon convergence. Training takes about one day for the Cityscapes dataset and two days for the

KITTI 360 dataset. The experiments are done using two Nvidia RTX 2080Ti.

C.3 Additional qualitative results

For the Cityscapes dataset, we train an additional model at higher resolution (i.e. 256×128 pixels) without changing any hyper-parameter. The results obtained with this model on two initial frames are shown in Figure C.1 and Figure C.2. These results are well in-line the the qualitative results reported in Chapter 4. We observe that the other methods are not able to move the object (see the red bounding boxes that indicate the initial position of the object). Indeed, the cars are either static, in Sheng *et al.* [103] and Sheng*, or blurry, in S. Sheng*. On the contrary, our approach is able to move the object and generates frames of good quality.

Figure C.4 and Figure C.5 instead show some additional visual results on KITTI 360.

Finally, Figure C.6 shows a qualitative example of the ablation study we performed in Section 4.3. The first row of this Figure shows a version of our network that does not model object interactions. By comparing the first and second rows of Figure C.6, we clearly see that, while the highlighted object correctly moves, all the other objects have unrealistic motions. This confirms the quantitative results about the importance of modelling the object interactions to have temporal consistent movements of different objects in the scene.

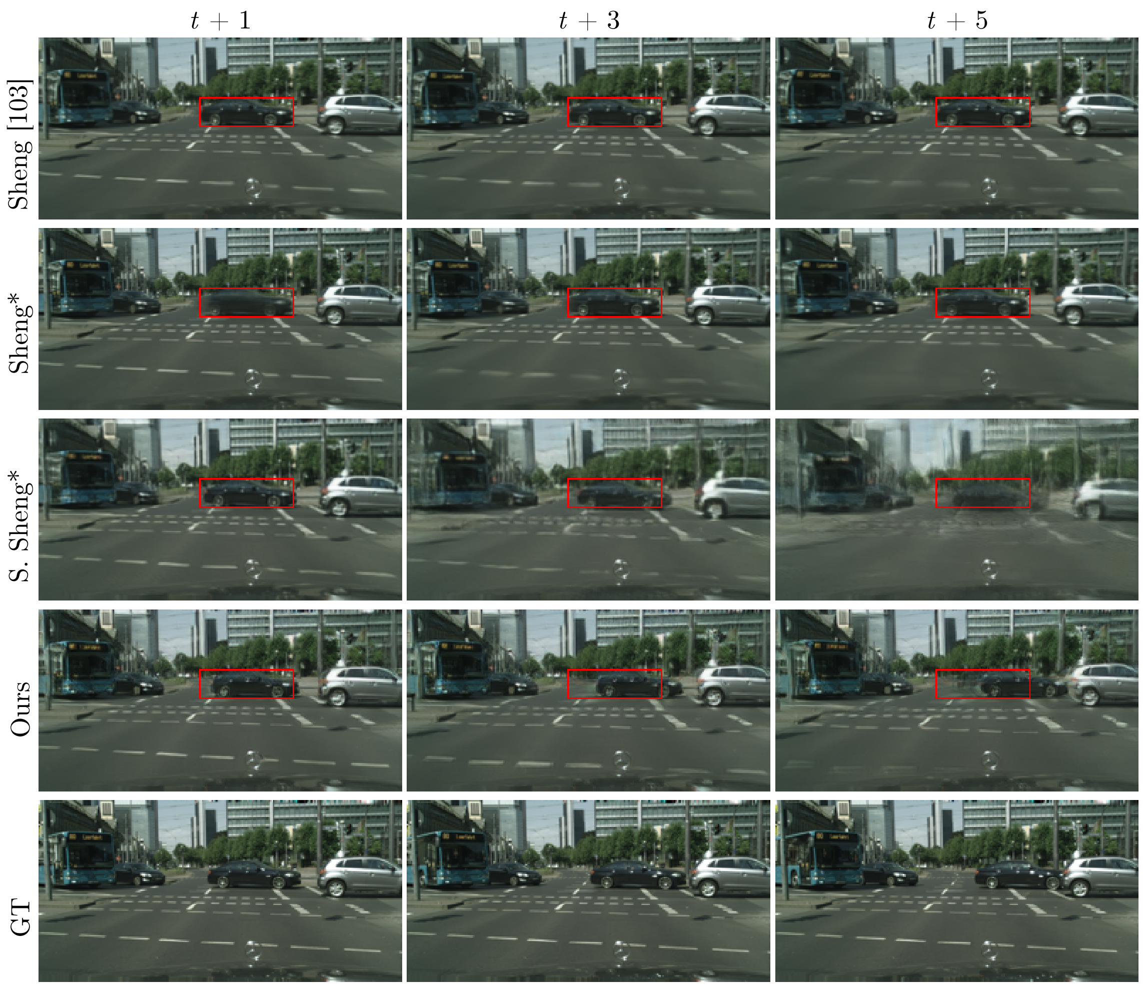

Figure C.1: Results of predicting the frames $t+1$, $t+3$, and $t+5$ on the Cityscapes dataset [14] with ground truth reference. On first three columns, we move the pedestrian near the semaphore to left. On the last three columns we move car crossing the street. The position of the moved object at $t=0$ is highlighted in red. Zoom for details.

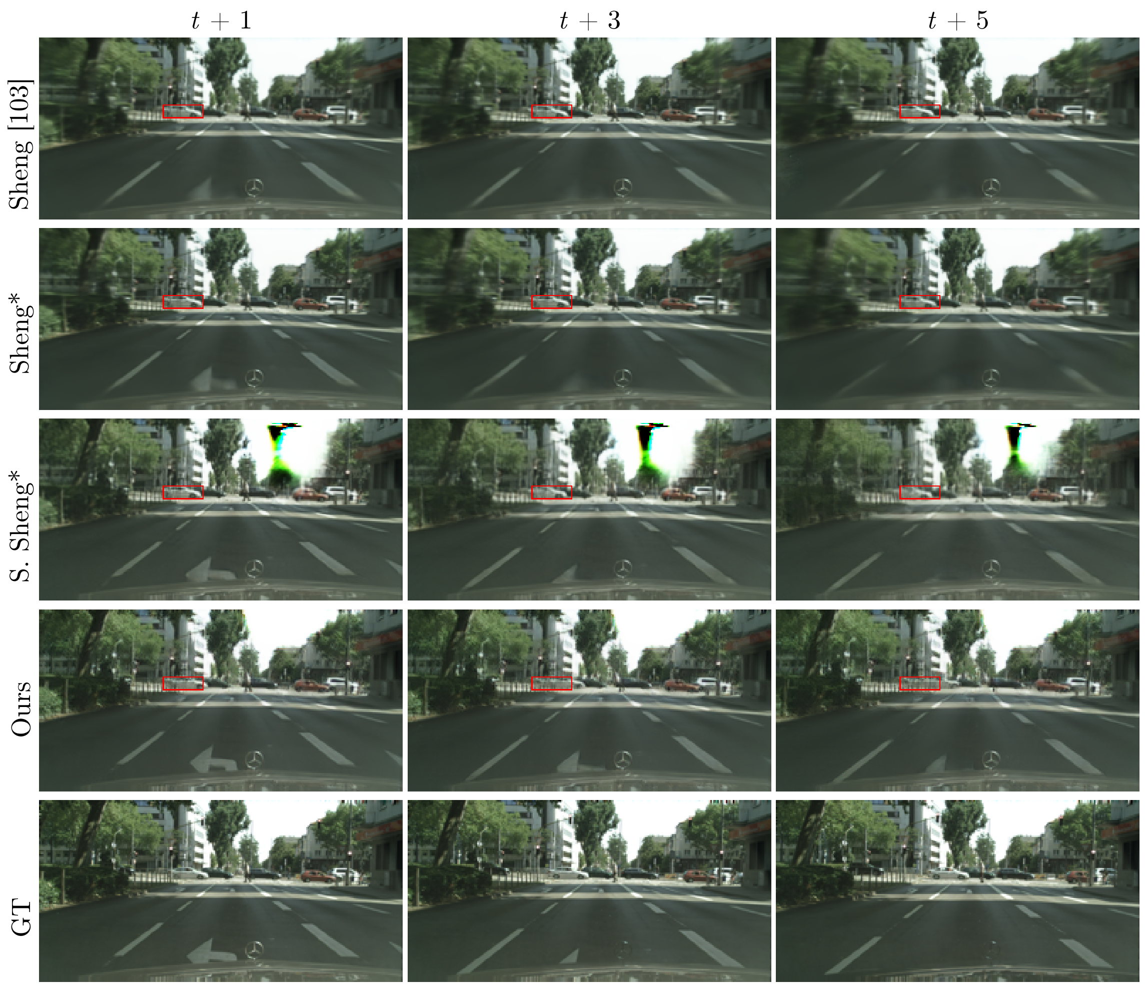

Figure C.2: Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the Cityscapes dataset [14] with ground truth reference. On first three columns, we move the pedestrian near the semaphore to left. On the last three columns we move car crossing the street. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details.

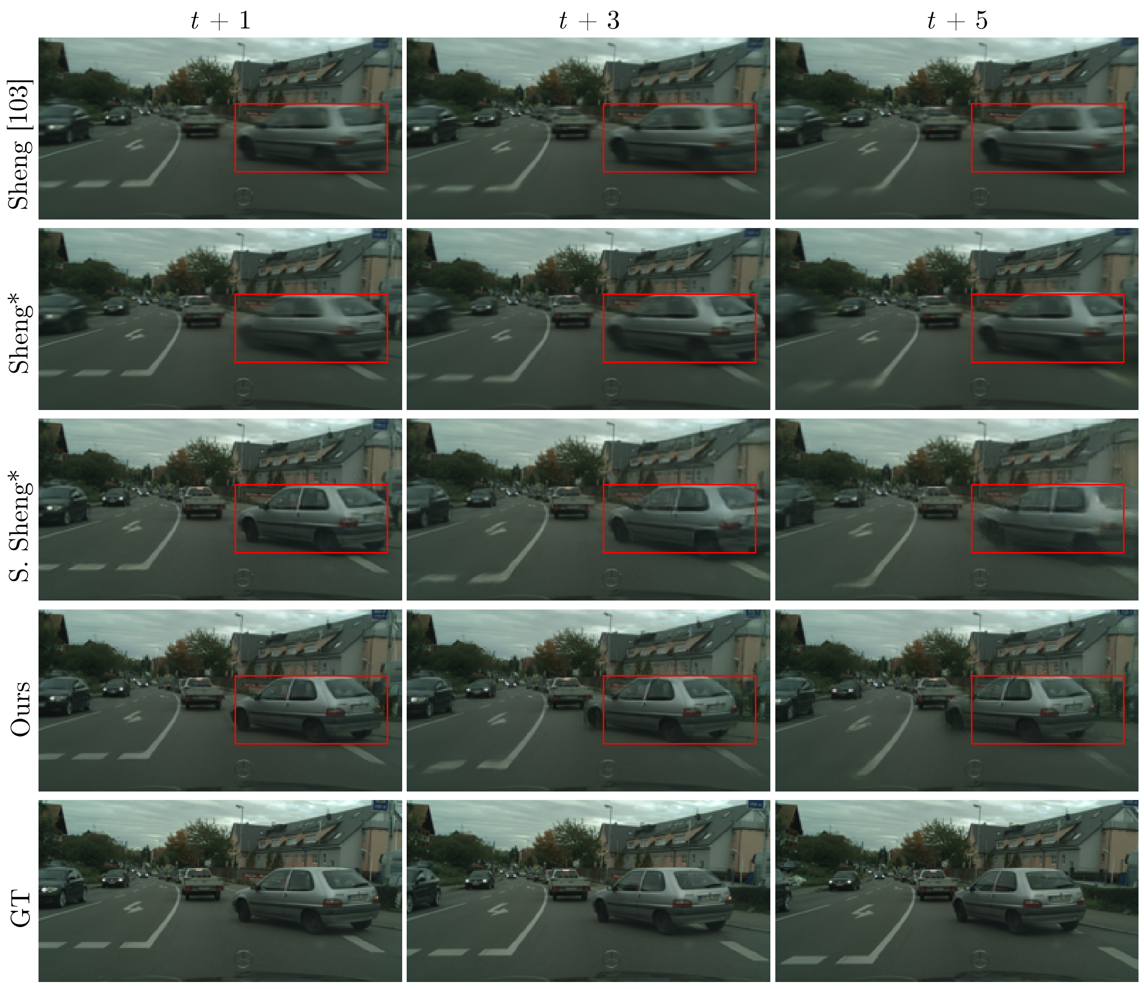

Figure C.3: Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the Cityscapes dataset [14] with ground truth reference. On first three columns, we move the pedestrian near the semaphore to left. On the last three columns we move car crossing the street. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details.

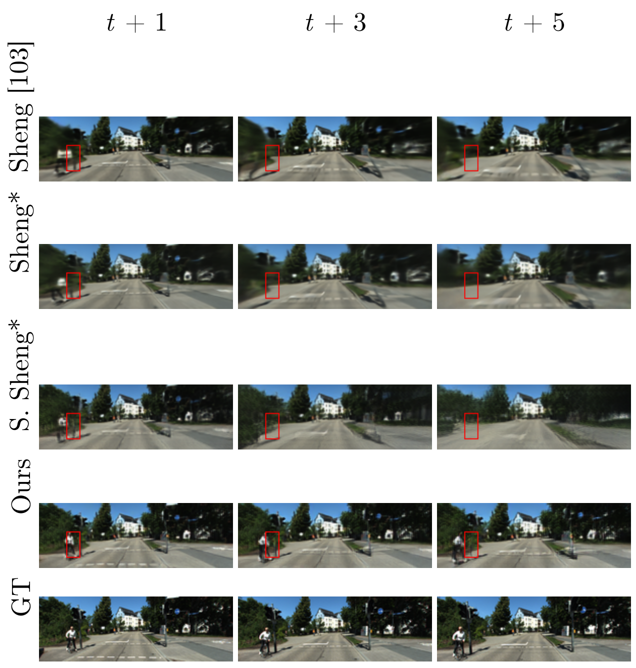

Figure C.4: Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the KITTI360 dataset [137]. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details.

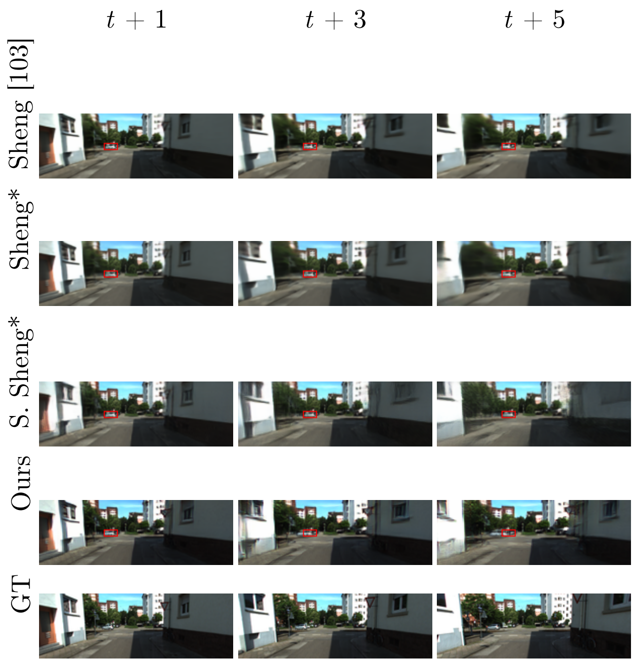

Figure C.5: Results of predicting the frames $t + 1$, $t + 3$, and $t + 5$ on the KITTI360 dataset [137]. The position of the moved object at $t = 0$ is highlighted in red. Zoom for details.

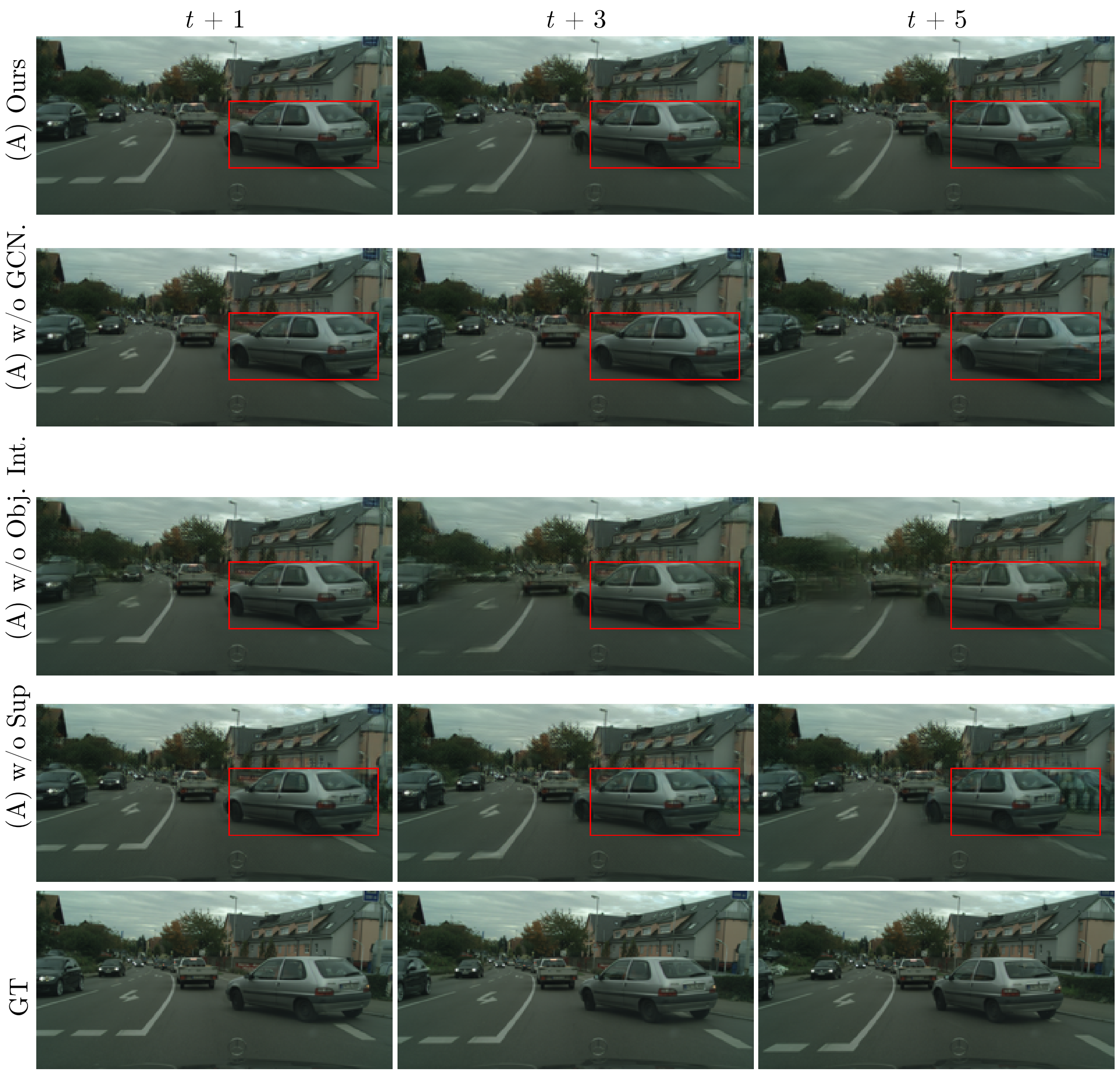

Figure C.6: Results of the ablation test on the Cityscapes dataset [14].

Bibliography

[1] Pierfrancesco Ardino, Yahui Liu, Elisa Ricci, Bruno Lepri, and Marco De Nadai. Semantic-guided inpainting network for complex urban scenes manipulation. In *ICPR*, 2021.

[2] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223, 2017.

[3] Samaneh Azadi, Catherine Olsson, Trevor Darrell, Ian Goodfellow, and Augustus Odena. Discriminator rejection sampling. In *International Conference on Learning Representations*, 2018.

[4] Lorenzo Berlincioni, Federico Becattini, Leonardo Galteri, Lorenzo Seidenari, and Alberto Del Bimbo. Road layout understanding by generative adversarial inpainting. In *Inpainting and Denoising Challenges*, pages 111–128. Springer, 2019.

[5] Sergio Casas, Cole Gulino, Simon Suo, Katie Luo, Renjie Liao, and Raquel Urtasun. Implicit latent variable model for scene-consistent motion forecasting. In *European Conference on Computer Vision*, pages 624–641. Springer, 2020.

[6] Caroline Chan, Shiry Ginosar, Tinghui Zhou, and Alexei A Efros. Everybody dance now. In *ICCV*, pages 5933–5942, 2019.

[7] Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6-7):772–799, 2008.

[8] Lele Chen, Ross K Maddox, Zhiyao Duan, and Chenliang Xu. Hierarchical cross-modal talking face generation with dynamic pixel-wise loss. In *CVPR*, pages 7832–7841, 2019.

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

[10] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[11] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *CVPR*, pages 12475–12485, 2020.

[12] Yunjey Choi, Youngjung Uh, Jaejun Yoo, and Jung-Woo Ha. Stargan v2: Diverse image synthesis for multiple domains. In *CVPR*, 2020.

[13] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[14] 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 *CVPR*, pages 3213–3223, 2016.

[15] Hanjun Dai, Yingtao Tian, Bo Dai, Steven Skiena, and Le Song. Syntax-directed variational autoencoder for molecule generation. In *Proceedings of the International Conference on Learning Representations*, 2018.

[16] Adnan Darwiche. Sdd: A new canonical representation of propositional knowledge bases. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[17] Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002.

[18] Nicola De Cao and Thomas Kipf. MolGAN: An implicit generative model for small molecular graphs. *arXiv preprint arXiv:1805.11973*, 2018.

[19] Ivan Donadello, Luciano Serafini, and Artur D'Avila Garcez. Logic tensor networks for semantic image interpretation. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, pages 1596–1602, 2017.

[20] Haoye Dong, Xiaodan Liang, Yixuan Zhang, Xujie Zhang, Xiaohui Shen, Zhenyu Xie, Bowen Wu, and Jian Yin. Fashion editing with adversarial parsing learning. In *CVPR*, pages 8120–8128, 2020.

[21] Haoye Dong, Xiaodan Liang, Yixuan Zhang, Xujie Zhang, Zhenyu Xie, Bowen Wu, Ziqi Zhang, Xiaohui Shen, and Jian Yin. Fashion editing with multi-scale attention normalization. *arXiv preprint arXiv:1906.00884*, 2019.

[22] Daan Fierens, Guy Van den Broeck, Joris Renkens, Dimitar Shterionov, Bernd Gutmann, Ingo Thon, Gerda Janssens, and Luc De Raedt. Inference and learning in probabilistic logic programs using weighted boolean

formulas. *Theory and Practice of Logic Programming*, 15(3):358–401, 2015.

[23] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, 2016.

[24] Chelsea Finn, Ian Goodfellow, and Sergey Levine. Unsupervised learning for physical interaction through video prediction. *arXiv preprint arXiv:1605.07157*, 2016.

[25] Jean-Yves Franceschi, Edouard Delasalles, Mickael Chen, Sylvain Lamprier, and P. Gallinari. Stochastic latent residual video prediction. *ArXiv*, abs/2002.09219, 2020.

[26] Tsu-Jui Fu, Licheng Yu, Ning Zhang, Cheng-Yang Fu, Jong-Chyi Su, William Yang Wang, and Sean Bell. Tell me what happened: Unifying text-guided video completion via multimodal masked video generation. *arXiv preprint arXiv:2211.12824*, 2022.

[27] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In *CVPR*, pages 2414–2423, 2016.

[28] Yalda Ghasemi, Heejin Jeong, Sung Ho Choi, Kyeong-Beom Park, and Jae Yeol Lee. Deep learning-based object detection in augmented reality: A systematic review. *Computers in Industry*, 139:103661, 2022.

[29] Francesco Giannini, Michelangelo Diligenti, Marco Gori, and Marco Maggini. On a convex logic fragment for learning and reasoning. *IEEE Transactions on Fuzzy Systems*, 27(7):1407–1416, 2018.

[30] Rafael Gómez-Bombarelli, Jennifer N Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[31] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[32] Gabriel Lima Guimaraes, Benjamin Sanchez-Lengeling, Carlos Outeiral, Pedro Luis Cunha Farias, and Alán Aspuru-Guzik. Objective-reinforced generative adversarial networks (ORGAN) for sequence generation models. *arXiv preprint arXiv:1705.10843*, 2017.

[33] Matthew Guzdial and Mark Riedl. Game level generation from gameplay videos. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[34] Xintong Han, Zuxuan Wu, Weilin Huang, Matthew R Scott, and Larry S Davis. Finet: Compatible and diverse fashion image inpainting. In *ICCV*, 2019.

[35] Zekun Hao, Xun Huang, and Serge Belongie. Controllable video generation with sparse trajectories. In *CVPR*, pages 7854–7863, 2018.

[36] Jiawei He, Andreas Lehrmann, Joseph Marino, Greg Mori, and Leonid Sigal. Probabilistic video generation using holistic attribute control. In *ECCV*, pages 452–467, 2018.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

[38] 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. In *NIPS*, pages 6626–6637, 2017.

[39] Yung-Han Ho, Chuan-Yuan Cho, Wen-Hsiao Peng, and Guo-Lun Jin. Sme-net: Sparse motion estimation for parametric video prediction through reinforcement learning. In *ICCV*, pages 10462–10470, 2019.

[40] Seunghoon Hong, Xinchen Yan, Thomas S Huang, and Honglak Lee. Learning hierarchical semantic image manipulation through structured representations. In *NIPS*, pages 2708–2718, 2018.

[41] Yaosi Hu, Chong Luo, and Zhenzhong Chen. Make it move: controllable image-to-video generation with text descriptions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 18219–18228, 2022.

[42] Zhiting Hu, Zichao Yang, Ruslan R Salakhutdinov, LIANHUI Qin, Xiaodan Liang, Haoye Dong, and Eric P Xing. Deep generative models with learnable knowledge constraints. In *Advances in Neural Information Processing Systems*, pages 10501–10512, 2018.

[43] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. Globally and locally consistent image completion. *ACM Transactions on Graphics (ToG)*, 36(4):1–14, 2017.

[44] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *CVPR*, pages 2462–2470, 2017.

[45] Adobe Inc. Photoshop: Now the world's most advanced AI application for creatives. `https://tinyurl.com/yzg97uaq`. Accessed: 2021-02-21.

[46] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[47] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. In *CVPR*, pages 1125–1134, 2017.

[48] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial transformer networks. *Advances in neural information processing systems*, 28, 2015.

[49] Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, José Miguel Hernández-Lobato, Richard E Turner, and Douglas Eck. Sequence tutor: Conservative fine-tuning of sequence generation models with kl-control. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1645–1654. JMLR. org, 2017.

[50] Youngjoo Jo and Jongyoul Park. Sc-fegan: Face editing generative adversarial network with user's sketch and color. In *ICCV*, pages 1745–1753, 2019.

[51] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *ECCV*. Springer, 2016.

[52] Tarun Kalluri, Girish Varma, Manmohan Chandraker, and CV Jawahar. Universal semi-supervised semantic segmentation. In *ICCV*, pages 5259–5270, 2019.

[53] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *ICLR'18*, 2018.

[54] Diederik Kingma and Max Welling. Auto-encoding variational Bayes. In *International Conference on Learning Representations*, 2014.

[55] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[56] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[57] Doga Kisa, Guy Van den Broeck, Arthur Choi, and Adnan Darwiche. Probabilistic sentential decision diagrams. In *Fourteenth International Conference on the Principles of Knowledge Representation and Reasoning*, 2014.

[58] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[59] Manoj Kumar, Mohammad Babaeizadeh, Dumitru Erhan, Chelsea Finn, Sergey Levine, Laurent Dinh, and Durk Kingma. Videoflow: A conditional flow-based model for stochastic video generation. In *ICLR*, 2020.

[60] Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1945–1954. JMLR. org, 2017.

[61] Alex X. Lee, Richard Zhang, Frederik Ebert, P. Abbeel, Chelsea Finn, and S. Levine. Stochastic adversarial video prediction. *ArXiv*, abs/1804.01523, 2018.

[62] Donghoon Lee, Sifei Liu, Jinwei Gu, Ming-Yu Liu, Ming-Hsuan Yang, and Jan Kautz. Context-aware synthesis and placement of object instances. In *NIPS*, 2018.

[63] Chun-Liang Li, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabás Póczos. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pages 2203–2213, 2017.

[64] Qizhu Li, Anurag Arnab, and Philip HS Torr. Weakly-and semi-supervised panoptic segmentation. In *ECCV*, pages 102–118, 2018.

[65] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Flow-grounded spatial-temporal video prediction from still images. In *ECCV*, pages 600–615, 2018.

[66] Yitong Li, Martin Min, Dinghan Shen, David Carlson, and Lawrence Carin. Video generation from text. In *AAAI*, volume 32, 2018.

[67] Xiaodan Liang, Lisa Lee, Wei Dai, and Eric P Xing. Dual motion gan for future-flow embedded video prediction. In *ICCV*, pages 1744–1752, 2017.

[68] Huan Ling, Karsten Kreis, Daiqing Li, Seung Wook Kim, Antonio Torralba, and Sanja Fidler. Editgan: High-precision semantic image editing. *Advances in Neural Information Processing Systems*, 34:16331–16345, 2021.

[69] Marco Lippi and Paolo Frasconi. Prediction of protein β-residue contacts by markov logic networks with grounding-specific weights. *Bioinformatics*, 25(18):2326–2333, 2009.

[70] Yahui Liu, Marco De Nadai, Deng Cai, Huayang Li, Xavier Alameda-Pineda, Nicu Sebe, and Bruno Lepri. Describe what to change: A text-

guided unsupervised image-to-image translation approach. In *MM'20*, page 1357–1365, New York, NY, USA, 2020. Association for Computing Machinery.

[71] Yahui Liu, Marco De Nadai, Jian Yao, Nicu Sebe, Bruno Lepri, and Xavier Alameda-Pineda. Gmm-unit: Unsupervised multi-domain and multi-modal image-to-image translation via attribute gaussian mixture modeling. *arXiv preprint arXiv:2003.06788*, 2020.

[72] Yahui Liu, Enver Sangineto, Yajing Chen, Linchao Bao, Haoxian Zhang, Nicu Sebe, Bruno Lepri, Wei Wang, and Marco De Nadai. Smoothing the disentangled latent style space for unsupervised image-to-image translation. In *CVPR*, pages 10785–10794, June 2021.

[73] Robin Manhaeve, Sebastijan Dumancic, Angelika Kimmig, Thomas Demeester, and Luc De Raedt. DeepProbLog: Neural probabilistic logic programming. In *Advances in Neural Information Processing Systems*, pages 3749–3759, 2018.

[74] Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *ICCV*, pages 2794–2802, 2017.

[75] Giuseppe Marra, Francesco Giannini, Michelangelo Diligenti, and Marco Gori. LYRICS: a General Interface Layer to Integrate AI and Deep Learning. *arXiv preprint arXiv:1903.07534*, 2019.

[76] Giuseppe Marra and Ondřej Kuželka. Neural markov logic networks. *arXiv preprint arXiv:1905.13462*, 2019.

[77] Michael Mathieu, Camille Couprie, and Yann LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[78] Lars Mescheder, Andreas Geiger, and Sebastian Nowozin. Which training methods for gans do actually converge? *arXiv preprint arXiv:1801.04406*, 2018.

[79] Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

[80] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Z Qureshi, and Mehran Ebrahimi. Edgeconnect: Generative image inpainting with adversarial edge learning. *arXiv preprint arXiv:1901.00212*, 2019.

[81] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulò, and Peter Kontschieder. The mapillary vistas dataset. In *ICCV*, 2017.

[82] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems*, pages 271–279, 2016.

[83] Augustus Odena, Vincent Dumoulin, and Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 1(10):e3, 2016.

[84] Xi Ouyang, Yu Cheng, Yifan Jiang, Chun-Liang Li, and Pan Zhou. Pedestrian-synthesis-gan: Generating pedestrian data in real scene and beyond. *arXiv preprint arXiv:1804.02047*, 2018.

[85] Junting Pan, Chengyu Wang, Xu Jia, Jing Shao, Lu Sheng, Junjie Yan, and Xiaogang Wang. Video generation from single semantic label map. In *CVPR*, pages 3733–3742, 2019.

[86] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *CVPR*, 2019.

[87] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *CVPR*, 2016.

[88] Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 689–690. IEEE, 2011.

[89] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[90] Raghunathan Ramakrishnan, Pavlo O Dral, Matthias Rupp, and O Anatole von Lilienfeld. Quantum chemistry structures and properties of 134 kilo molecules. *Scientific Data*, 1, 2014.

[91] Fitsum A Reda, Guilin Liu, Kevin J Shih, Robert Kirby, Jon Barker, David Tarjan, Andrew Tao, and Bryan Catanzaro. Sdc-net: Video prediction using spatially-displaced convolution. In *ECCV*, pages 718–733, 2018.

[92] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[93] Yurui Ren, Xiaoming Yu, Ruonan Zhang, Thomas H Li, Shan Liu, and Ge Li. Structureflow: Image inpainting via structure-aware appearance flow. In *ICCV*, 2019.

[94] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. In *Advances in Neural Information Processing Systems*, pages 3788–3800, 2017.

[95] Lars Ruddigkeit, Ruud van Deursen, Lorenz C. Blum, and Jean-Louis Reymond. Enumeration of 166 billion organic small molecules in the

chemical universe database gdb-17. *Journal of Chemical Information and Modeling*, 52(11):2864–2875, 2012. PMID: 23088335.

[96] Masaki Saito, Eiichi Matsumoto, and Shunta Saito. Temporal generative adversarial nets with singular value clipping. In *Proceedings of the IEEE international conference on computer vision*, pages 2830–2839, 2017.

[97] Mehdi SM Sajjadi, Bernhard Scholkopf, and Michael Hirsch. Enhancenet: Single image super-resolution through automated texture synthesis. In *ICCV*, 2017.

[98] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

[99] Bidisha Samanta, DE Abir, Gourhari Jana, Pratim Kumar Chattaraj, Niloy Ganguly, and Manuel Gomez Rodriguez. Nevae: A deep generative model for molecular graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1110–1117, 2019.

[100] Bidisha Samanta, Abir De, Niloy Ganguly, and Manuel Gomez-Rodriguez. Designing random graph models using variational autoencoders with applications to chemical design. *arXiv preprint arXiv:1802.05283*, 2018.

[101] Ari Seff, Wenda Zhou, Farhan Damani, Abigail Doyle, and Ryan P Adams. Discrete object generation with reversible inductive construction. *arXiv preprint arXiv:1907.08268*, 2019.

[102] Yujun Shen, Jinjin Gu, Xiaoou Tang, and Bolei Zhou. Interpreting the latent space of gans for semantic face editing. In *CVPR*, pages 9243–9252, 2020.

[103] Lu Sheng, Junting Pan, Jiaming Guo, Jing Shao, and Chen Change Loy. High-quality video generation from static structural annotations. *International Journal of Computer Vision*, 128:2552–2569, 2020.

[104] Wenzhe Shi, Jose Caballero, Ferenc Huszar, Johannes Totz, Andrew P Aitken, Rob Bishop, Daniel Rueckert, and Zehan Wang. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *CVPR*, pages 1874–1883, 2016.

[105] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of big data*, 6(1):1–48, 2019.

[106] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. Animating arbitrary objects via deep motion transfer. In *CVPR*, 2019.

[107] Aliaksandr Siarohin, Stéphane Lathuilière, Sergey Tulyakov, Elisa Ricci, and Nicu Sebe. First order motion model for image animation. *Neurips*, 2019.

[108] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[109] Sam Snodgrass and Santiago Ontanón. Controllable procedural content generation via constrained multi-dimensional markov chain sampling. In *IJCAI*, pages 780–786, 2016.

[110] Yuhang Song, Chao Yang, Yeji Shen, Peng Wang, Qin Huang, and C-C Jay Kuo. Spg-net: Segmentation prediction and guidance network for image inpainting. In *bmvc*, 2018.

[111] Adam Summerville and Michael Mateas. Super mario as a string: Platformer level generation via lstms. *arXiv preprint arXiv:1603.00930*, 2016.

[112] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. Procedural content generation via machine learning (pcgml). *IEEE Transactions on Games*, 10(3):257–270, 2018.

[113] Adam James Summerville, Sam Snodgrass, Michael Mateas, and Santiago Ontanón. The vglc: The video game level corpus. *arXiv preprint arXiv:1606.07487*, 2016.

[114] Wei Sun and Tianfu Wu. Learning layout and style reconfigurable gans for controllable image synthesis. *arXiv preprint arXiv:2003.11571*, 2020.

[115] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *CVPR*, 2016.

[116] Jerry Talton, Lingfeng Yang, Ranjitha Kumar, Maxine Lim, Noah Goodman, and Radomír Měch. Learning design patterns with bayesian grammar induction. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 63–74. ACM, 2012.

[117] Ruben Rodriguez Torrado, Ahmed Khalifa, Michael Cerny Green, Niels Justesen, Sebastian Risi, and Julian Togelius. Bootstrapping conditional gans for video game level generation. *arXiv preprint arXiv:1910.01603*, 2019.

[118] Yukitaka Tsuchiya, Takahiro Itazuri, Ryota Natsume, Shintaro Yamamoto, Takuya Kato, and Shigeo Morishima. Generating video from single image and sound. In *CVPR Workshops*, pages 17–20, 2019.

[119] Sergey Tulyakov, Ming-Yu Liu, Xiaodong Yang, and Jan Kautz. Mocogan: Decomposing motion and content for video generation. In *CVPR*, pages 1526–1535, 2018.

[120] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv:1607.08022*, 2016.

[121] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *CVPR*, pages 6924–6932, 2017.

[122] Thomas Unterthiner, Sjoerd van Steenkiste, Karol Kurach, Raphaël Marinier, Marcin Michalski, and Sylvain Gelly. FVD: A new metric for video generation. *arXiv preprint arXiv:1812.01717*, 2019.

[123] Guy Van den Broeck, Nima Taghipour, Wannes Meert, Jesse Davis, and Luc De Raedt. Lifted probabilistic inference by first-order knowledge compilation. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence*, pages 2178–2185. AAAI Press/International Joint Conferences on Artificial Intelligence; Menlo . . . , 2011.

[124] Vladimir N Vapnik and A Ya Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pages 11–30. Springer, 2015.

[125] Girish Varma, Anbumani Subramanian, Anoop Namboodiri, Manmohan Chandraker, and CV Jawahar. Idd: A dataset for exploring problems of autonomous navigation in unconstrained environments. In *WACV*. IEEE, 2019.

[126] Ruben Villegas, Jimei Yang, Yuliang Zou, Sungryull Sohn, Xunyu Lin, and Honglak Lee. Learning to generate long-term future via hierarchical prediction. In *ICML*, pages 3560–3569. PMLR, 2017.

[127] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 221–228. ACM, 2018.

[128] Carl Vondrick, Hamed Pirsiavash, and Antonio Torralba. Anticipating the future by watching unlabeled video. *arXiv preprint arXiv:1504.08023*, 2, 2015.

[129] Carl Vondrick and Antonio Torralba. Generating the future with adversarial transformers. In *CVPR*, pages 1020–1028, 2017.

[130] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Guilin Liu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. Video-to-video synthesis. *arXiv preprint arXiv:1808.06601*, 2018.

[131] 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 *CVPR*, pages 8798–8807, 2018.

[132] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[133] Olivia Wiles, A Koepke, and Andrew Zisserman. X2face: A network for controlling face generation using images, audio, and pose codes. In *ECCV*, pages 670–686, 2018.

[134] Yue Wu, Rongrong Gao, Jaesik Park, and Qifeng Chen. Future video synthesis with object motion prediction. In *CVPR*, pages 5539–5548, 2020.

[135] Yue Wu, Qiang Wen, and Qifeng Chen. Optimizing video prediction via video frame interpolation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 17814–17823, 2022.

[136] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[137] Jun Xie, Martin Kiefel, Ming-Ting Sun, and Andreas Geiger. Semantic instance annotation of street scenes by 3d to 2d label transfer. In *CVPR*, 2016.

[138] Wei Xiong, Jiahui Yu, Zhe Lin, Jimei Yang, Xin Lu, Connelly Barnes, and Jiebo Luo. Foreground-aware image inpainting. In *CVPR*, pages 5840–5848, 2019.

[139] Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International Conference on Machine Learning*, pages 5498–5507, 2018.

[140] Yexiang Xue and Willem-Jan van Hoeve. Embedding decision diagrams into generative adversarial networks. In *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 616–632. Springer, 2019.

[141] Yufei Ye, Maneesh Singh, Abhinav Gupta, and Shubham Tulsiani. Compositional video prediction. In *ICCV*, pages 10353–10362, 2019.

[142] Fisher Yu, Vladlen Koltun, and Thomas Funkhouser. Dilated residual networks. In *CVPR*, pages 472–480, 2017.

[143] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative image inpainting with contextual attention. In *CVPR*, 2018.

[144] Tao Yu, Zongyu Guo, Xin Jin, Shilin Wu, Zhibo Chen, Weiping Li, Zhizheng Zhang, and Sen Liu. Region normalization for image inpainting. In *AAAI*, 2020.

[145] Fangneng Zhan, Yingchen Yu, Rongliang Wu, Jiahui Zhang, and Shijian Lu. Multimodal image synthesis and editing: A survey. *arXiv preprint arXiv:2112.13592*, 2021.

[146] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

[147] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5907–5915, 2017.

[148] Jiangning Zhang, Chao Xu, Liang Liu, Mengmeng Wang, Xia Wu, Yong Liu, and Yunliang Jiang. Dtvnet: Dynamic time-lapse video generation via single still image. In *ECCV*, pages 300–315. Springer, 2020.

[149] Bo Zhao, Lili Meng, Weidong Yin, and Leonid Sigal. Image generation from layout. In *CVPR*, pages 8584–8593, 2019.

[150] Chuanxia Zheng, Tat-Jen Cham, and Jianfei Cai. Pluralistic image completion. In *CVPR*, pages 1438–1447, 2019.

[151] Yang Zhou, Xintong Han, Eli Shechtman, Jose Echevarria, Evangelos Kalogerakis, and Dingzeyu Li. Makelttalk: speaker-aware talking-head animation. *ACM Transactions on Graphics (TOG)*, 39(6):1–15, 2020.

[152] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2223–2232, 2017.