Simulations for Cognitive Vision

Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften

vorgelegt beim Fachbereich Informatik and Mathematik
der Johann Wolfang Goethe-Universität
in Frankfurt am Main

von

VSR Veeravasarapu
aus Vadali, India

Frankfurt am Main, 2018
D30

vom Fachbereich Informatik und Mathematik der
Johann Wolfgang Goethe-Universität Frankfurt am Main als Dissertation angenommen.

Dekan: Prof. Dr. Andreas Bernig

1. Gutachter: Prof. Dr. Visvanathan Ramesh

2. Gutachter: Prof. Dr. Constantin Rothkopf

Datum der Disputation:

The greatest challenge to any thinker is stating the problem in a way that will allow a solution

Bertrand Russell –

Acknowledgements

I owe an enormous intellectual debt to my supervisor Prof. Visvanathan Ramesh who introduced me to systems engineering principles and several fields of computer vision domain. His uncanny abilities to find weak links in a chain of reasoning always amazed me and inspired me to view the problems from a different perspective. His constant encouragement and guidance made this work possible. The knowledge and valuable suggestions given by him provided the foundation for the work presented in this thesis. I am also grateful for the support and encouragement of Dr. Constantin Rothkopf, my secondary guide who gave me valuable feedback, suggestions, and inputs. I should be thankful to him for the cooperation in writing the camera-ready version of my research article which was accepted at CVPR-2017.

I also want to thank the team at Center for Cognition and Computation lab, especially, Dr. Kishore Konda and Rudra Hota, for lengthy discussions and their support of all things technical. I must also thank Tobias Weis, Martin Mundt, Christian Becker and others in the administrative team for providing a wonderful research environment at the lab and maintaining easily accessible computation resources.

I greatly acknowledge the funding towards my Ph.D. from the German Federal Ministry of Education and Research (BMBF) in projects 01GQ0840 and 01GQ0841 (BFNT Frankfurt) and Continental automotive GmbH.

Abstract

Due to the resurrection of data-hungry models (such as deep convolutional neural nets), there is an increasing demand for large-scale labeled datasets and benchmarks in the computer vision fields (CV). However, collecting real data across diverse scene contexts along with high-quality annotations is often expensive and time-consuming, especially for detailed pixel-level label prediction tasks such as semantic segmentation, etc. To address the scarcity of real-world training sets, recent works have proposed the use of computer graphics (CG) generated data to train and/or characterize performance of modern CV systems. CG based virtual worlds provide easy access to ground truth annotations and control over scene states. Most of these works utilized training data simulated from video games and pre-designed virtual environments and demonstrated promising results. However, little effort has been devoted to the systematic generation of massive quantities of sufficiently complex synthetic scenes for training scene understanding algorithms. In this work, we develop a full pipeline for simulating large-scale datasets along with per-pixel ground truth information. Our simulation pipeline constitutes of mainly two components: (a) a stochastic scene generative model that automatically synthesizes traffic scene layouts by using marked point processes coupled with 3D CAD objects and factor potentials, (b) an annotated-image rendering tool that renders the sampled 3D scene as RGB image with a chosen rendering method along with pixel-level annotations such as semantic labels, depth, surface normals etc. This pipeline is capable of automatically generating and rendering a potentially infinite variety of outdoor traffic scenes that can be used to train convolutional neural nets (CNN).

However, several recent works, including our own initial experiments demonstrated that the CV models that are trained naively on simulated data lack generalization capabilities to real-world scenes. This opens up several fundamental questions about what is it lacking in simulated data compared to real data and how to use it effectively. Furthermore, there has been a long debate since 1980's on the usefulness of CG generated data for tuning CV systems. Primarily, the impact of modeling errors and computational rendering approximations, due to various choices in the rendering pipeline, on trained CV systems generalization performance is still not clear. In this thesis, we take a case study in the context of traffic scenarios to

empirically analyze the performance degradations when CV systems trained with virtual data are transferred to real data. We first explore system performance tradeoffs due to the choice of the rendering engine (e.g., Lambertian shader (LS), ray-tracing (RT), and Monte-Carlo path tracing (MCPT)) and their parameters. A CNN architecture, DeepLab, that performs semantic segmentation, is chosen as the CV system being evaluated. In our case study, involving traffic scenes, a CNN trained with CG data samples generated with photorealistic rendering methods (such as RT or MCPT), shows already a reasonably good performance on real-world testing data from CityScapes benchmark. Use of samples from an elementary rendering method, i.e., LS, degraded the performance of CNN by nearly 20%. This result conveys that training data must be photorealistic enough for better generalizability of the trained CNN models. Furthermore, the use of physics-based MCPT rendering improved the performance by 6% but at the cost of more than three times the rendering time. This MCPT generated dataset when augmented with just 10% of real-world training data from CityScapes dataset, the performance levels achieved are comparable to that of training CNN with the complete CityScapes dataset.

The next aspect we study in the thesis involves the impact of choice of parameter settings of scene generation model on the generalization performance of CNN models trained with the generated data. Towards this end, we first propose an algorithm to estimate our scene generation model parameters given an unlabeled real world dataset from the target domain. This unsupervised tuning approach utilizes the concept of generative adversarial training, which aims at adapting the generative model by measuring the discrepancy between generated and real data in terms of their separability in the space of a deep discriminatively-trained classifier. Our method involves an iterative estimation of the posterior density of prior distributions for the generative graphical model used in the simulation. Initially, we assume uniform distributions as priors over parameters of a scene described by our generative graphical model. As iterations proceed the uniform prior distributions are updated sequentially to distributions for the simulation model parameters that leads to simulated data with statistics that are closer to the distributions of the unlabeled target data.

In order to quantify the impact of parameter settings of the scene generation, we compare the generalization of the CNN models trained separately on virtual datasets that are generated with the scene parameters before and after tuning to target domain's data. We demonstrate better generalization capabilities of DCNN models by tuning the scene generation (using the proposed adversarial tuning method) to two real-world benchmark datasets (CityScapes and CamVid). We obtained performance improvements by 2.28% and 3.14% in mIOU metrics between the CNN models trained on simulated sets prepared from the scene generation models, before and after tuning, on CityScapes and CamVid datasets respectively. Moreover, we observe that utility of simulated data generated from the tuned models further reduces the amount of labeled real data by nearly 1% (to reach the performance levels on par with that of complete CityScapes data).

Although the utility of physics-based renderer and tuning the parameters of scene generation has resulted in improved generalization performance, we needed several "labeled" real samples (at least 9% of CityScapes in the context of experiments) to achieve the performance levels on par with that of full real-world training. However, this 9% data (i.e., more than 300 images) still requires a lot of human efforts to label at pixel-level, for instance, optical flow and intrinsic images, etc. By motivating the fact that unlabeled data is abundant in real-world, we next explore the use of unlabeled real data in a Multi-Task Learning (MTL) framework to reduce the domain shift of the feature representations that are learned while training the vision models on labeled simulated data. We present a novel MTL based framework that uses an auxiliary task on "unlabeled" real data to regularize the processes of training on labeled simulated data to learn feature representations that are more generalizable to reality. Hence, the proposed MTL framework simultaneously trains two CNNs: one for semantic segmentation on labeled simulated data and the other for an auxiliary self-supervised task on unlabeled real data. Since the geometric context-based features are relatively less biased compared to appearance models, we choose to solve an in-painting task as the algorithms have to understand the geometric context of the entire image. However, training with traditional reconstruction losses such as L2-norm might produce blurry results. We surmount this problem by using an adversarial discriminator to help the designed in-painting network to produce high-quality predictions. Our experiments prove that the MTL approach can learn feature representations that are well generalizable to real-world scenarios. In specific, MTL improves the generalization of semantic segmentation model by at least 11%. Interestingly, the impact of rendering engine has become insignificant on the CNN models trained in our MTL framework, while tuning of simulators still seems to important as data generated w/o tuning degraded the performance nearly by 7%. This experimental evidence empirically supports our claim that our MTL training process emphasizes to transfer geometric context rather than appearance specific features.

Zusammenfassung

Begründet in der Auferstehung daten-hungriger Modelle (z.B. tiefe neuronale Netze) steigt die Nachfrage nach großangelegten annotierten Datensätzen - vor allem in den verschiedenen Feldern des künstlichen Sehens. Die Sammlung von realen Daten über verschiedene Szenenkontexte hinweg sowie qualitativ hochwertige Annotationen sind jedoch oft kostspielig und zeitaufwendig, insbesondere für detaillierte, Pixel-genaue Vorhersagen wie z.B. semantische Segmentierung. Um dem Mangel an realen Trainings-Datensätzen zu begegnen, haben jüngste Arbeiten die Verwendung von durch Computergrafik (CG) generierten Daten vorgeschlagen, um moderne CV-Systeme zu trainieren und deren Performanz zu charakterisieren. CG-basierte virtuelle Welten bieten einfachen Zugriff auf "ground-truth" Annotationen und erlauben volle Kontrolle über die Zustände von Szenarien. Die meisten dieser Arbeiten verwendeten Trainingsdaten, die aus Videospielen und vorgefertigten virtuellen Umgebungen simuliert wurden, und zeigten vielversprechende Ergebnisse.

Es wurden jedoch nur wenige Anstrengungen unternommen, um systematisch große Mengen an ausreichend komplexen synthetischen Szenen zu erzeugen, um damit Algorithmen zum Szenen-Verständnis zu trainieren. In dieser Arbeit entwickeln wir eine vollständige Pipeline zur Simulation großer Datensätze mit Pixel-genauen "ground-truth"-Annotationen. Unsere Simulations-Pipeline besteht im Wesentlichen aus zwei Komponenten: (a) ein stochastisches Szenengenerationsmodell, das Verkehrsszenen-Layouts unter Verwendung von markierten Punktprozessen in Verbindung mit 3D-CAD-Objekten und Faktorpotentialen automatisch synthetisiert, (b) ein Rendering-Tool für annotierte Bilder, das die gesampelte 3D-Szene als RGB-Bild mit einem gewählten Renderingverfahren zusammen mit Pixel-genauen Annotationen wie semantischen Labels, Tiefenwerten, Oberflächennormalen, usw. darsgetellt. Diese Pipeline ist in der Lage, automatisch eine potenziell unendliche Vielfalt von Verkehrsszenen zu erzeugen und zu rendern, die zum Trainieren von "faltenden neuronalen Netzen" (convolutional neural network - CNN) verwendet werden knnen.

Mehrere aktuelle Arbeiten, darunter unsere eigenen vorläufigen Experimente, zeigten jedoch, dass die CV-Modelle, die naiv auf simulierten Daten trainiert werden, durch mangelnde Generalisierung nur schlecht auf reale Szenen anwendbar sind. Dies wirft einige grundlegende Fragen auf: Was fehlt den simulierten Daten im Vergleich

zu realen Daten, und wie knnen diese effektiver genutzt werden? Darüber hinaus gibt es seit den 1980er Jahren eine lange Debatte über den Nutzen von CG-generierten Daten für das Tuning von CV-Systemen.

In erster Linie sind die Auswirkungen von Modellierungsfehlern und rechnerischen Rendering-Approximationen aufgrund verschiedener Entscheidungen in der Rendering-Pipeline auf die Generalisierungsleistung trainierter CV-Systeme noch unklar. In dieser Arbeit analysieren wir empirisch an einer Fallstudie im Rahmen von Verkehrsszenarien die Performanz-Verluste eines CV-Systems, das mit virtuellen Daten trainiert, und auf echten Daten angewendet wird. Wir untersuchen den Einfluss der Wahl des Rendering-Verfahrens (z.B. Lambertian Shader (LS), Ray-Tracing (RT) und Monte-Carlo Path Tracing (MCPT)) und deren Parameter auf die System-Performanz. Als CV-System, das ausgewertet wird, wird eine CNN-Architektur gewählt (DeepLab), die eine semantische Segmentierung durchführt.

Unsere Fallstudie zeigt, dass ein auf photo-realistischen gerenderten (z.B. RT oder MCPT) CG-Daten trainiertes CNN bereits eine annehmbare Performanz auf echten Test-Daten des CityScapes-Datensatzes liefert. Die Verwendung von Daten generiert durch elementare Renderingverfahren (LS) verschlechterte die Leistung des CNNs um fast 20%. Dieses Ergebnis zeigt, dass die Trainingsdaten photorealistisch genug sein müssen, um eine bessere Generalisierung der trainierten CNN-Modelle zu erreichen. Darüber hinaus verbesserte der Einsatz von physikalisch basiertem MCPT-Rendering die Leistung um 6%, allerdings auf Kosten von mehr als dem Dreifachen der Renderzeit. Durch Anreicherung des mit MCPT generierten Datensatzes mit nur 10% der realen Trainingsdaten aus dem CityScapes-Datensatz wurden Performanz-Niveaus erreicht, die vergleichbar sind mit dem eines CNNs, welches auf dem kompletten echten CityScapes-Datensatz trainiert wurde.

Der nächste Aspekt, den wir in der Arbeit untersuchen, betrifft die Auswirkungen der Wahl der Parametereinstellungen des Szenengenerierungsmodells auf die Generalisierungsleistung von CNN-Modellen, die mit den generierten Daten trainiert wurden. Zu diesem Zweck schlagen wir zunächst einen Algorithmus vor, um unsere Modellparameter für die Szenengenerierung zu schätzen, wenn ein nicht annotierter Datensatz realer Daten aus der Zieldomäne verwendet wird. Dieser unüberwachte Tuning-Ansatz nutzt das Konzept des generativen gegnerischen Trainings, das darauf abzielt, das generative Modell anzupassen, indem es die Diskrepanz zwischen erzeugten und realen Daten im Hinblick auf ihre Trennbarkeit im Raum eines tiefen, diskriminierend trainierten Klassifikators misst. Unser Verfahren beinhaltet eine iterative Schätzung der a-posteriori Dichte von a-priori Verteilungen für das in der

Simulation verwendete generative grafische Modell. Zunächst gehen wir von uniformen Verteilungen als a-priori Verteilungen über Parameter einer Szene aus, die durch unser generatives grafisches Modell beschrieben werden. Im Laufe der Iterationen werden die uniformen a-priori Verteilungen sequentiell zu den Verteilungen für die Simulationsmodellparameter aktualisiert, was zu simulierten Daten führt, deren Statistiken näher an den Verteilungen der nicht annotierten Zieldaten liegen.

Um die Auswirkungen der Parameter der Szenengenerierung zu quantifizieren, vergleichen wir die Generalisierung von CNN-Modellen, die separat auf virtuellen Datensätzen trainiert wurden, welche mit Szenenparametern vor und nach dem Tuning auf die Daten der Zieldomäne erzeugt werden. Wir demonstrieren bessere Generalisierungs-Fähigkeiten von DCNN-Modellen, indem wir die Szenengenerierung (mit der vorgeschlagenen gegnerischen Tuning-Methode) auf zwei reale Benchmark-Datensätze (CityScapes und CamVid) abstimmen. Wir erzielten Leistungssteigerungen von 2,28% bzw. 3,14% bei den mIOU-Metriken zwischen den CNN-Modellen, die auf simulierten Sets trainiert wurden, die aus den Szenengenerierungsmodellen vor und nach dem Tuning auf CityScapes- und CamVid-Datensätzen erstellt wurden. Darüber hinaus stellen wir fest, dass der Nutzen von simulierten Daten, die aus den getunten Modellen generiert werden, die Menge der annotierten Echtdaten weiter um fast 1% reduziert (um das Leistungsniveau auf dem Niveau der vollständigen CityScapes-Daten zu erreichen).

Obwohl der Nutzen des physikalisch basierten Renderers und das Abstimmen der Parameter der Szenengenerierung zu einer verbesserten Generalisierungsleistung geführt hat, bentigten wir mehrere annotierte reale Daten (mindestens 9% von CityScapes im Rahmen von Experimenten), um vergleichbare Performanz zu Modellen zu erreichen, welche auf dem vollständigen realen Datensatz trainiert wurden. Diese 9% Daten (d.h. mehr als 300 Bilder) erfordern jedoch immer noch viel menschliche Anstrengung, um diese Pixel-genau zu annotieren, z.B. mit optischem Fluss und intrinsischen Bildern, etc. Durch die Motivation der Tatsache, dass nicht-annotierte reale Daten reichlich vorhanden sind, untersuchen wir als nächstes die Verwendung nicht-annotierter realer Daten in einem Multi-Task Learning (MTL)-Framework, um die Domänenverschiebung der Feature-Repräsentationen zu reduzieren, die beim Training der CV-Modelle auf annotierten simulierten Daten erlernt werden.

Wir präsentieren ein neuartiges MTL-basiertes Framework, das eine Hilfsaufgabe für "nicht-annotierte" reale Daten verwendet, um die Prozesse des Trainings auf annotierten simulierten Daten zu regularisieren, um Feature Repräsentationen zu lernen, die besser auf reale Daten generalisieren. Daher trainiert das vorgeschlagene MTL-Framework gleichzeitig zwei CNNs: eines für die semantische Segmentierung auf annotierten simulierten Daten und das andere für eine zusätzliche selbstüberwachte Aufgabe auf nicht-annotierten realen Daten.

Da die geometrischen kontextbasierten Merkmale im Vergleich zu Erscheinungsmodellen relativ geringe systematische Fehler aufweisen, entscheiden wir uns für die Lsung einer In-Painting-Aufgabe, da die Algorithmen den geometrischen Kontext des gesamten Bildes verstehen müssen. Allerdings kann ein Training mit traditionellen Rekonstruktionsverlusten wie der L2-Norm zu verschwommenen Ergebnissen führen. Wir überwinden dieses Problem, indem wir einen gegnerischen Diskriminator einsetzen, der dem entworfenen In-Painting-Netzwerk hilft, qualitativ hochwertige Vorhersagen zu treffen. Unsere Experimente beweisen, dass der MTL-Ansatz Feature-Repräsentationen lernen kann, die gut auf reale Szenarien generalisieren. Insbesondere verbessert MTL die Generalisierung des semantischen Segmentierungsmodells um mindestens 11%. Interessanterweise ist der Einfluss des Rendering-Engine auf die in unserem MTL-Framework trainierten CNN-Modelle unbedeutend geworden, während das Tuning von Simulatoren immer noch zu wichtig erscheint, da die ohne Tuning erzeugten Daten die Performanz um fast 7% verschlechterten. Diese experimentellen Beweise stützen empirisch unsere Behauptung, dass unser MTL-Trainings prozess den Schwerpunkt auf die übertragung geometrischer Zusammenhänge und nicht auf optische Besonderheiten legt.

Contents

List of Figures

1 Introduction

Recent advances in computational paradigms and scalable optimization schemes have revolutionized the disciplines of computer vision and computer graphics. These advances along with the availability of large scale datasets have facilitated the use of machine learning based models with a large number of tunable parameters (for example, deep convolutional neural networks) for nearly all forms of vision applications. Due to the complex nonlinear nature of these models, conventional performance modeling or parameter tuning methods (such as uncertainty or distribution propagation etc.) can become intractable computations for modern vision systems. Training with large scale *diverse data*, followed by extensive testing provides a practical way to assure the required performance. Furthermore, both training and testing are being done in supervised fashion that requires high quality ground truth information for all the samples in the dataset. In practice, the processes of such data collection might be costly and laborious, especially in situations where the data must cover a wide range of scenarios (for instance, images from different seasons etc.) and training labels require pixel-level annotations. For instance, CityScapes dataset [16] is one of the most commonly used datasets for traffic scene semantic segmentation. It consists of roughly 20000 RGB images in total, out of which only 3475 images are annotated with pixel-level labels while remaining are unlabeled or partially labeled. This clearly is insufficient for the supervised training of modern computer vision methods, especially deeper neural networks. Moreover, the manual labeling of per-pixel ground truth information is time consuming and error-prone, limiting both its quantity and accuracy. Hence, lack of good diverse data with corresponding ground truth information would be a major bottleneck in the vision system design processes.

Parallel to vision fields, computer graphics (CG) has seen major advances in 3D CAD modeling and photorealistic renderings, thanks to video gaming and movie industries. Modern rendering methods are able to produce visually realistic and physically plausible images and videos. This has drawn the attention of vision researchers to utilize CG generated data to overcome the problems of scarcity of real annotated data. Hence, several recent works proposed the utility of simulated datasets to train and/or diagnose vision systems. Utilizing CG generated data in the design processes may provide several additional advantages. Parametric control over the virtual scene states allows us to produce the diverse scenarios, including the most frequent states and rarely occurring events. Access to ground truth representations paves a way to train computer vision (CV) systems/models in a supervised manner, even for the tasks in which real world ground truth is not measurable, for instance, optical flow and intrinsic images etc. The trained systems can be validated extensively even in what-if scenarios. This can be done in very early phases even when the physical host platforms are not available. Moreover, the trade-offs between system's performance and hardware setup can be estimated in virtual environments. This will avoid costly redesigns later in the process.

1.1 Motivations and Problem Statements

Although CG generated data promise control over scene states and access to ground truth information, it is not trivial to generate synthetic data in large quantities with sufficient diversity. This is due to (a) lack of stochastic generative models for complex and diverse 3D scene configurations, and (b) the high computational cost of generating photo-realistic renderings at a massive scale. Hence, most of existing works use data from video games or manually designed virtual environments. Unfortunately, these sets may not provide full control and access to the 3D scenes. For example, the work of [63] had to develop a semi-automated labeling process which requires at least one manual-seed per a mesh object, to annotate the data generated from a video game known as GTA-V. Although it reduces time and efforts needed for manual annotation processes, it is still not entirely automated as the renderer does not have access to the semantic information of its scene elements.

We believe that a fully automatic and controllable simulation workbench can enable the vision system design process in many ways. Hence, in this work we develop a complete pipeline for synthesizing large scale urban scene layouts via 1) sampling scene instances from a probabilistic 3D scene model, and 2) performing graphics rendering of the scenes (with flexible choice of rendering methods) along with detailed pixel-level ground-truth that serves as input training data for Machine Learning. Our data simulation pipeline has the following characteristics:

- By leveraging freely available online 3D CAD repositories, we develop a probabilistic scene generative model that can generate 3D scene states with common layout constraints such as mutual spatial exclusion, alignment and other inter-object relations etc.

- We also develop an image rendering tool that can render RGB images along with required ground truth information for a 3D scene state sampled from the above probabilistic scene model. The tool incorporates with several rendering methods ranging from classical to modern physically realistic rendering engines to be used in a plug-and-play manner.

The utility of CG-based simulations in the design process of CV systems is not new and has been a long-debated issue since 1980's. Many researchers expressed apparently diverging opinions about the role of CG for experimental vision. Early modeling efforts and their assumptions underlying vision algorithms significantly overlap with the ones that CG based simulations use (for example, Lambertian reflection [52] and Dichromatic scattering models [51]. Thus, the CV community had been skeptical of using CG for learning as the data generated was assumed to be ideal or near-ideal inputs to CV algorithms and noise models are unrealistic [77]. However, modern CG rendering methods are in a state that where they can simulate visually realistic images and videos by using physically inspired lighting computations. Also, modern CV architectures are based on deep hierarchical networks, which learn the features automatically from the given training data. Hence, the utility of CG generated data to train or diagnose modern CV systems has gained a renewed attention, and, thus, in recent years, we have seen the increased use of synthetic datasets.

Having said that modern CG rendering can produce photorealistic training data, several recent works [80, 85, 25, 65, 63, 68] reported that the modern CV models trained "only" on these virtual datasets are lacking generalization capabilities on real-world data. This again raises several fundamental questions such as: 1) What properties are lacking in simulated data when compared to real-world data? , 2) When is simulated data useful for training vision algorithms that can perform adequately on real-world data ? CG based simulations are based on several mathematical and computational approximations. How do these approximations impact the generalizability of the CV systems trained on virtual data? These questions are not yet systematically addressed. This work address this space.

Our experimental studies attempt to answer these questions by taking fraffic scene understanding as an application domain. More specifically, we utilize the simulation tools developed in this work to examine the following aspects:

- impact of rendering fidelity (modeling errors and computational approximations in rendering),

- impact of scene generation parameter settings on the generalization performance of a vision system trained on simulated data,
- a set of design practices that encourage vision algorithm to learn domain-invariant features that are generalizable to real images.

1.2 Contributions and Outline

The contributions of this thesis include:

- **An Unifying Perspective (Chapter 2)**: We start by reviewing some works in the literature that shed light on the role of CG based simulations for experimental vision. Some of these works expressed apparently diverging opinions about the utility of simulated data in the design process of computer vision systems. We place their results in the context of systems characterization methodology outlined in the 90's and note that insights derived from simulations can be qualitative or quantitative depending on the degree of fidelity of models used in simulation and the nature of the question posed by the experimenter. This perspective explains the apparently diverging opinions expressed in the literature about the utility of simulated data for experimental vision.
- **Data Rendering (Chapter 3)**: To render image data along with required ground truth information, we devise a data rendering tool based on an open source graphics platform, *Blender.* The tool is integrated with several rendering methods ranging from classical to modern rendering engines and annotation shaders in a manner that facilitates plug and play with several data simulation methods. The details are discussed in Chapter 3. We also revisit the pipeline of graphics rendering from a vision researcher's perspective to try to point out what is it lacking compared to real world image formation physics. We later use this tool to render a set of scenes with different rendering methods and parameters and try to quantify their impact of the generalization performance of the trained model in Chapter 5.
- **Stochastic Scene Generation (Chapter 4)**: Training a deep network requires a large number of images with sufficient sample diversity in the training phase. Manual design of complex 3D scene states that too in large quantities with sufficient diversity may require many designers and man-hours. Hence, we propose to use a probabilistic scene generative model that is based on marked point processes coupled with predownloaded 3D CAD models and

factor potentials to encode inter-object constraints such mutual exclusion and alignment etc.

- **Impact of Rendering Fidelity (Chapter 5)**: It is not yet clear that how the rendering approximations (modeling errors and computational approximations in rendering method) impact the generalization performance of a vision model trained on the simulated data, especially in modern computer vision systems that utilize deep learning. In Chapter 5, we address this question in the context of traffic scene semantic understanding by comparing the generalization performance of a deep learning framework trained on simulated data rendered with different choices of rendering engines and their parameters.

- **Adversarially Tuned Scene Generation (Chapter 6)**: Our parametric scene generative model facilitates the freedom of selecting probabilistic distributions of scene parameters such as lighting and weather, etc. In the above chapters, we use uniform or Gaussian distributions of parameters in their permissible ranges and generate the samples. However, we believe that utilizing the distributions that are closer to the ones of target real world data might improve the CV system's performance on the testing data coming from the same target domain. Hence, in this chapter we propose an unsupervised tuning methodology to estimate the prior distributions for our scene generative models by using a real-world dataset via the use of adversarial training concepts [3].

- **Impact of Scene Generation Parameters (Chapter 7)**: We analyze the impact of parameter choices of scene generative model on the system's generalization to real-world target data. In this chapter, we consider two publicly available real-world benchmark datasets as target sets and show how our tuned scene generation processes can improve the generalization capabilities of trained models to target setting.

- **Learning to Transfer Geometric Context from Virtual to Reality (Chapter 8)**: In the above chapters, We focussed on making simulated data statistically similar to real testing data and thus improving generalization performances of trained models. In this chapter, we focus on unsupervised visual learning approach that encourages training on simulated data to learn the features are more realistic and generalizable to real data. Our approach is based on Multi Task Learning that trains for semantic segmentaion on simulated data along with an auxiliary task on real data in self-supervised manner. Our experiments in Chapter 8 demonstrate that the generalization performance has been significantly improved by using this MTL framework.

- **Discussion and Conclusions (Chapter 9)**: We finally conclude by consolidating all experimental findings, and provide recommentations, in practice, for generation and utilization of CG data to train a DCNN. We also point out a few weaknesses of our work and mention possible future extensions.

1.3 Publications

- **V. S. R. Veeravasarapu**, C. A. Rothkopf, V. Ramesh. Model-driven simulations for computer vision. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017. Santa Rosa, USA.
- **V. S. R. Veeravasarapu**, C. A. Rothkopf, V. Ramesh. Adversarially tuned scene generation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. Honolulu, Hawaii.

2 Background

This chapter presents a general overview of the works that utilize CG generated data for different purposes in the development cycle of modern vision algorithms, including training, inference, performance modeling and improvements etc.

2.1 Synthetic Imagery for Vision

Due to scarcity of real world annotated datasets, CG generated image datasets have recently been a source of training data for many applications of CV such as semantic segmentation, object detection, pedestrian detection, 2D pose estimation, 3D human pose estimation, optical flow, action recognition and view point estimation etc.

Haltakov et al [31] used a racing video game to generate data with different pixel level groundtruth such as optical flow, semantic class labels and depth information. Similarly, another work [63] used recent and advanced video game (known as GTA-V) to generate the large scale data. However, the annotation process is not automated as the video game does not provide access to 3D scene state information. They proposed a semi automated annoatation process based on the communication between the game and graphics hardware. Another set of works [65, 64] used manually designed or procedurally generated 3D CAD city environments to generate the data randomly placing the object and camera in the scenes. For instance the works of synthea and action data. Virtual KITTI [25] is another interesting work which simulates the data that looks like KITTI benchmark dataset. All these works demostrated that virtual data when combined with a few samples of real world data can achieve state-of-the-art performances. Recently, [24] used a combination of real-data and synthetic objects to train a DCNN and provide empirical evidence of the usefulness of such training for real world settings.

Although some prior work demonstrates the potential of synthetic imagery to advance computer vision research, some other works reported some negative results when using synthetic data. For instance, experiments of [77] concluded that simulations are not useful for tuning optical flow systems. The work [49] used carefully designed indoor scene models (parameters) and advanced rendering algorithms to synthesize very realistic sensory data and concluded that the optical flow error spatial statistics are different between simulated and real data. On the other hand, the work [9] showed that motion models and local spatial statistics, crucial for optic flow estimation, match with reality. Hence, they argued that the artificial animated (Sintel) data could be used to design and tune the flow estimators even though the data is not photo-realistic. This issue of photorealism for virtual-world based training has a long debated issue in the literature. However, a systematic study about photorealism of simulations for vision is not yet been done due to lack of controllable and configurable data simulation platforms.

2.2 Diverging Opinions and Unifying Perspective

The main focus of the above mentioned works is in demonstration of the utility or lack of utility of simulation for vision systems design and the emphasis is on evaluation of the system as a black-box in an application context. Still, the degree of effectiveness of graphics rendered data for vision system design is an open question. A few discussions[1][2] on public platforms clearly convey the confusion that the vision community has towards using graphics for learning. These apparent divergent conclusions can be explained away by the perspective that utility of CG for CV depends on closeness of simulation models to reality and invariant natures of feature transforms in the system to be trained/validated.

Systems Characterization Perspective

In Systems science and engineering fields, **Simulations** have often been used to evaluate and understand the system as a whole and also the interactions between its parts/components. This kind of study is often referred as *Modeling and Simulation* (M&S). Simulations help in characterizing the performance of the system under certain situations without actually testing it in real life. The role of modeling and simulation in systems characterization (or performance characterization) of computer vision systems engineering was articulated in the early 90s [32, 61]. In these works, algorithms are viewed as statistical estimators and performance modeling of an algorithm is viewed as a process of establishing the correspondence between the deviations of an algorithms expected output as a function of the input signal parameters, perturbation model parameters, and algorithm tuning constants. These works advocate a perspective to view the simulation process as part of a meta-analysis stage for characterizing systems performance. Essentially, the emphasis is on how

[1]www.quora.com/How-useful-are-massive-virtual-game-environments-for-training-AI

[2]www.researchgate.net/post/Synthetic-datasets-vs-real-images-for-computer-vision-algorithm-evaluation2

sources of uncertainty propagate through a chosen system as a function of both input probability distributions and parameter choices used in the system. In this paradigm, a separate model validation stage is used for comparing theoretical models to real-world data. Simulation is used for statistical validation of correctness of software implementation as well as gauging performance limits of systems.

Although Computer Graphics is motivated by well established theories, the end rendering model that drives computations (for instance, Monte-carlo path tracing), generally incorporates modeling assumptions and approximated computations that are not theoretically motivated. The results of graphic simulations, therefore, do not automatically come with a stamp of approval that carries the full faith and credit of the vision models and their statistics/analytics. Hence, constant concern with uncertainty and error should be given, just same as real world data. Moreover, due to the fact that we may not be able to model the context very close to (unknown) real world models, it obviously reflects as dataset-shift problem in rendered data.

Transferability of the conclusions/models from simulations to reality depends on many factors and variables in the experimental pipeline that include

- Scene generative models ($\hat{P}(\theta_W)$) used,
- Rendering techniques ($\hat{G}(\theta_G)$) used to generate observations,
- Real world target data (D_r),
- Algorithm or System being evaluated (S)
- Criterion Function or metric for evaluation (L)

where θ_W denotes the generative model parameters governing the world scene and θ_G represents the parameters governing the computer graphics rendering choices.

Ideally, the scene distribution (world model) has to be propagated through the components $\hat{G}$ and S to get uncertainties in the conclusions/models that are being transferred to the real world testing. This is often called as white-box performance characterization. In a model-driven design paradigm, the structure of the system can be motivated from invariance arguments and designed modules have specific semantics in context (e.g. Mann [47], Mallat [8], Greiffenhagen [29] etc). In this setting the system S is designed by choice of invariant modules and systems analysis that explicitly account for contextual models and tasks. However, Model validation of generative models is a challenging task in the real-world. Modular specification of the world model (e.g. geometry, albedo, color, dynamics, environment), allows one to learn the model parameters separately and then use the physics-based simulator.

Modular specification naturally allows the use of a lesser number of samples. However, experimental design and sampling needs to be done carefully to learn accurate prior distributions. The strategy for validation of a model based design is (An example of model validation is given in Greiffenhagen et al [29]).

The rationale for modern deep CV architectures is currently being studied extensively by theorists. However, a holistic systems analysis linking physics based rendering methods with modern CV algorithm architectures is challenging. Hence, we resort to a black-box but rigorous analysis in which we consider rendering engines and vision systems (deep networks) as black-boxes. The combinations of ($\hat{P}$, $\hat{G}$, D_r, S, L) form a joint space of consideration for the experiment. Our choices in this thesis work can be seen as a specific example from this space. However, our experimental procedure can be extended to other choices with minimal changes.

Due to the explosions of big data (for example ImageNet) and renaissance of data hungry models (such as deep networks), the focus on modeling errors has been overshadowed with the excellence of deep models trained on large scale data. The performance of the deep models depends on scale and diversity of the training dataset. Moreover, with the adoption of deep networks from image-level classification to pixel-level prediction tasks, the cost of getting labeled data (especially for pixel level annotations) is increasing and becoming laborious. Hence, several recent attempts [80, 85, 25, 65, 63, 68] has been reported to utilize CG based simulations to train deep networks and demonstrated that amount of real world data required for training can be reduced. However, these works bypassed the more involved task of generative modeling by rendering data from video game environments and existing 3D CAD models. Moreover, they have taken a fundamental assumption that modern CG rendering methods simulate realistic images/videos. Hence, the impact of model choices and approximations in rendering pipeline is largely unexplored. In this work, we address this space with an approach inspired from systems characterization.

3 Graphics Rendering

In this chapter, we first provide a brief review of the major components in a CG-based rendering pipeline. We also provide the details of our data rendering tool along with the rendering methods used in our experiments in the following chapters.

3.1 A Brief Review of Computer Graphics

The major focus of Computer graphics is to simulate very realistic images/videos. This process mainly constitutes of two phases: 3D Modeling and Rendering. Both the phases have seen major advances in the last decades, thus, had a significant impact on many types of media and has revolutionized the fields of video games, animation, movies, and advertising etc.

3.1.1 3D modeling

CG pipeline starts with the process of developing a mathematical representation of three-dimensional (3D) virtual scene. This includes defining: 1) the geometrical structure of the objects in the scene, 2) a set of materials ribing the appearance of the objects, 3) the specification of the light sources in the scene, and 4) a virtual camera model. The geometry is often specified in terms of discretized rendering primitives such as triangles describing surface patches. The materials define how light interacts with surfaces and participating media. Finally, the scene is illuminated using one or several light sources, and the composition of the rendered frame is defined by introducing a virtual camera.

Due to high demands in video games and movie industry, a large number of softwares and tools have been developed to simplify the phase of 3D modeling. For example, Cinema 4D, Maya, 3DS max, Blender, LightWave, and Modo etc. All these tools provide programmable shaders for most of the object materials, light sources, weather scatterting effects and also camera models etc. Furthermore, these tools have opened doors for large-scale online collections of 3D CAD models to share across communities. For instance, please see Figure 3.1 that shows some samples of 3D CAD models for "Car" object category which are available on an online CAD

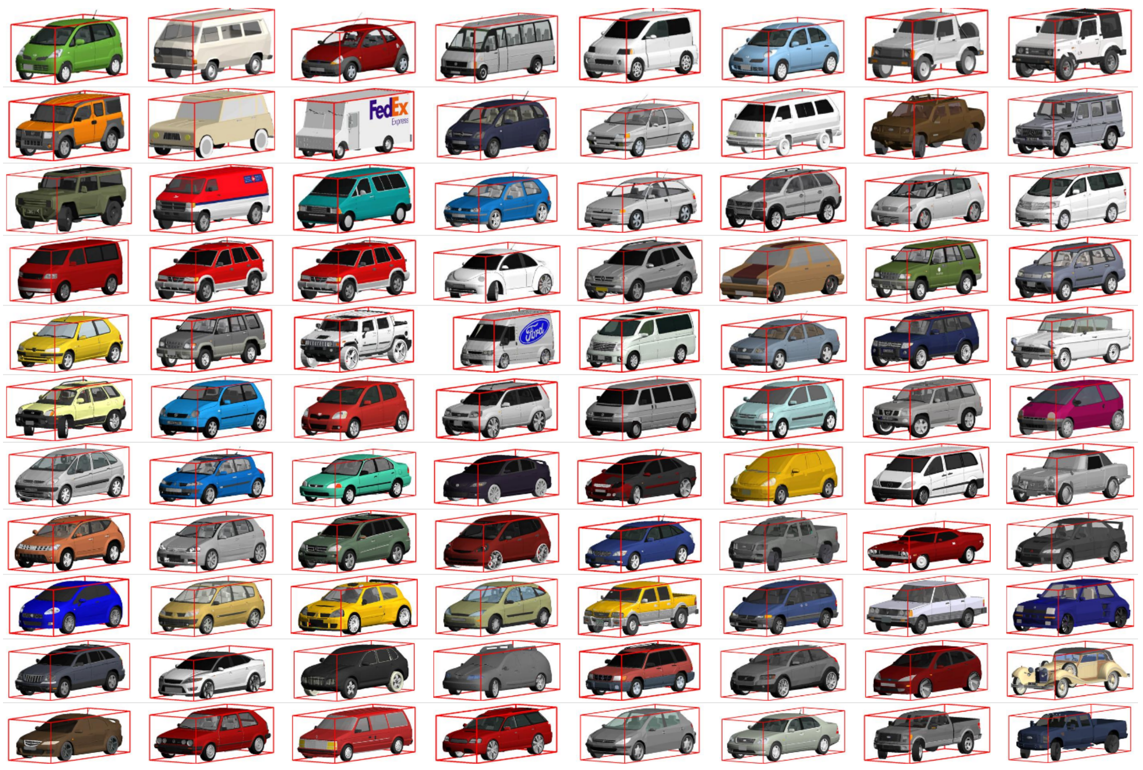


Figure 3.1: A few 3D CAD vehicle models downloaded from Google's 3D warehouse. These CAD models are used in [23] for 3D object detection.

repository. However manual designing the 3D virtual scenes is a laborious process, hence, we devise an automated scene generation protocol based probabilistic models and publicly available CAD object meshes. The details of this stochastic scene generation will be discussed in the next chapter.

3.1.2 Rendering

Once 3D scene state is generated either manually or automatically, next step in CG pipeline is rendering process that synthesizes 2D image/video(s) from the created 3D scene. This can be compared to taking a photo or filming the scene with a camera in real life. Ideally any rendering method aim to simulate the physics of light propagation through the entire 3D scene. Light propagation in a scene, from light source to camera, mainly involves three phenomena: (a) surface reflectance, (b) media scattering and (b) camera transfer.

3.1.2.1 Surface reflectance

The light interaction with surfaces [39] is explained mathematically as follows:

$$L_r(x,\omega_o) = L_e(x,\omega_o) + \int_H \rho_{bd}(x,\omega_o,\omega_i)L_i(x,\omega_i)\cos\theta_i d\sigma_{\omega_i} \tag{3.1}$$

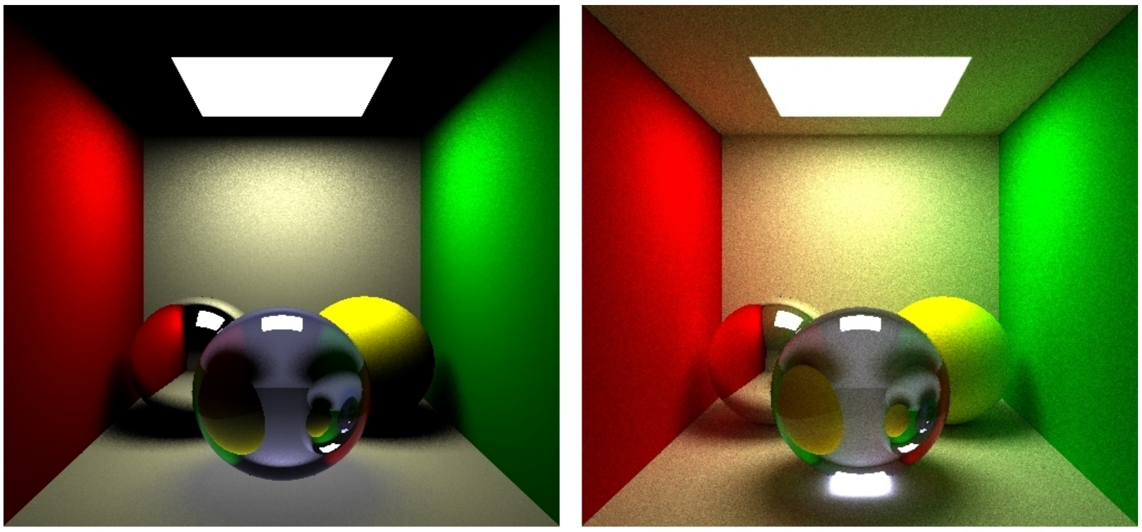


Figure 3.2: Raytracing (RT) vs Monte-carlo path tracing (MCPT) [1]. Several rendering algorithms have been proposed based on different approximations with different end purposes. For instance, RT (left) is favoured for real time rendering (video games) and MCPT (right) is favoured for physics-based renderings (movies or scientific visualizations). How does this rendering fidelity impact the tranferability of computer vision models to reality, when they are trained on rendering data?

where $L_r(x, \omega_o)$ and $L_e(x, \omega_o)$ are the reflected and emitted light in the direction ω_o at a surface point x. $L_i(x, \omega_i)$ is incident light which has to integrated over solid angle σ_{ω_i} of a infinitesimal hemisphere H centered at x. ρ_{bd} is BRDF (Bi-directional reflective distribution function) or BTF (Bi-directional transmission function), a function of incident (ω_i) and outgoing (ω_o) directions. θ_i is the angle between incident light ray and surface normal. BRDF/BTF [50] is a characteristic for a surface material that defines apperance of the material. Several graphics platforms such as Blender etc. provide these functions for a wide variety of materials that are commmonly found in nature. Equation 3.1 involve a continous integration of dot product (b/w incident light and BRDF) over a hemisphere H centered at surface point. This is analytically intractable to compute. Some methods ignore this integration and considers only one light ray that is coming directly from light source. These methods are oftenly refered as local illumination models or direct lighting based methods. For instance, Lambertian shader and Specular shader etc. The other set of methods either introduce mathmatical simplifications for real time rendering or use numerical ways to solve the continuous integration such as Monte-Carlo methods.

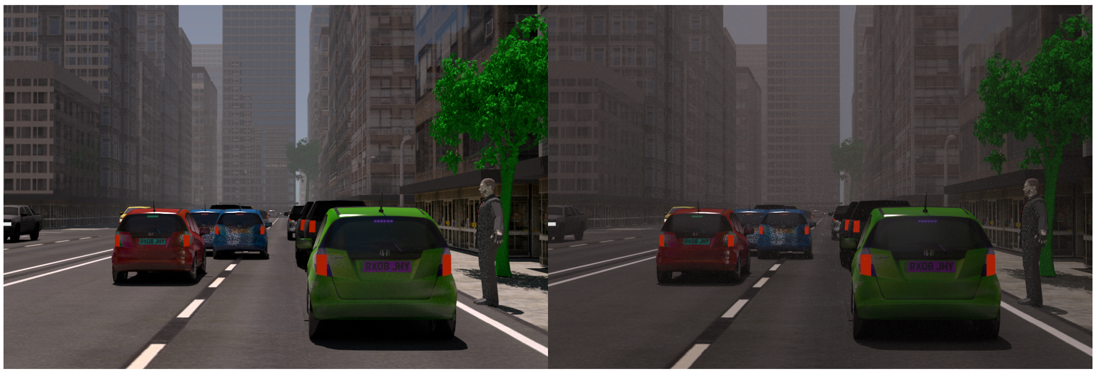

(a) volumetric weighting (b) volume scattering

Figure 3.3: Mathematically simplified and physically based Fog effects

3.1.2.2 Media scattering

When simulating images under weather such fog etc., reflected light (L_r) from the surfaces has to travel through medium/atmosphere before it hits camera. In this phase, it undergoes three kinds of phenomena: absorption, scattering and emission. This is formulated by the integral light transport equation as follows,

$$E(y) = \begin{aligned} &\underbrace{L_r(x,\omega_o)e^{-\int_x^y k_t(u)du}}_{E_{al}\text{: attenuated light}} + \underbrace{\int_x^y k_a(u)L_e(u)e^{-\int_u^y k_t(u)du}du}_{E_{el}\text{: emitted light}} \\ &+ \underbrace{\int_x^y \int_S \frac{k_s(u)}{4\pi} e^{\int_u^y k_t(u)du} L_r(x,\omega_i)p(\omega_o,\omega_i)d\sigma_{\omega_i}du}_{E_{ss}\text{: single scattering}} \\ &+ \underbrace{\int_x^y \int_S \frac{k_s(u)}{4\pi} e^{\int_u^y k_t(u)du} L_m(u,\omega_i)p(\omega_o,\omega_i)d\sigma_{\omega_i}du}_{E_{ms}\text{: multiple scattering or medium radiance}} \end{aligned} \tag{3.2}$$

where k_a, k_t and k_s are absorption, extinction and scattering coefficients of medium. $L_m(u)$ is medium radiance due to multiple scattering and in integrated over sphere S and along the light travel. Due to emission and inscattering, radiance increases because of light impinging on a point u that is scattered into the viewing direction. The spatial distribution of the scattered light is modeled by the phase function $p(\omega_o, \omega_i)$ and different phase functions (such as Mie [17] etc.) have been proposed and applied to simulated polluted sky, haze, clouds, and fog etc. This equation is also analytically intractable to solve as it involves double continuous integration. One class of methods, physics based methods, hire Monte-Carlo rendering methods. For instance Monte-Carlo light transport methods. These methods often use Schlick phase functions [37] that are parameterized by anisotropy and particle density. These are proven to be well approximations for theoretical functions and well suited for Monte Carlo rendering methods. Please see Figure 3.5c for scenes with fog rendered with physics based renderer. These methods are popularly used in movies and scientific visualizations etc. The other class of methods mainly simulate the appearance of weather effects with simple mathematical approximations. These are termed as appearance driven methods. For instance in Figure 3.5b, fog effect is added to an image with a post processing shader which uses a volumetric formula based depth buffer and color of the fog. These are popularly used in the situations where rendering should be done in real time, for example video games etc.

3.1.2.3 Camera transfer

Most of graphics people use *OpenGL* pinhole camera and its projection matrix to compute image intensities from camera irradiance (light incident on camera from the scene). However, the appearance of an image also depends on the type of sensor used and light transferring functionaties in it. CG developers **rarely consider sensor models and their influences on the rendered images**. We believe that simulated images, although realistic to the naked eye, might not be well suited for machine vision applications unless camera imperfections are considered, which include chromatic aberration, vignetting, lens distortion, sensor noise etc. Light (camera irradiance) in a classical camera undergoes through several steps in the camera before it stored as a pixel intensity [74]. Final digitized pixel is given by,

$$\hat{z} = f(a \int_0^t E(y)dt + N_s + N_{c_1} + b) + N_{c_2} \tag{3.3}$$

where $\hat{z}$ is the image intensity, f is the camera response function; a and b are the parameters of while balance module which linearly stretches the received camera irradiance $E(y)$ in the exposure time t. N_s, N_{c_1} and N_{c_2} are factors due to shot noise, thermal noise and quantization noise. **An accurate implementation of these phenomenon is mostly not available**. However, most of the graphics platform such as Blender provide image processing based filters to add these effects while post-processing the rendered image. Some of these effects are as shown in Figure 3.4.

3.1.3 Rendering engines

Several different, and often specialized, rendering methods have been developed for different end-goals. These range from the direct lighting based rendering to more advanced techniques such as: *Ray tracing* or *Monte-Carlo path tracing* etc. Rendering may take from fractions of a second to hours for a single image/frame,

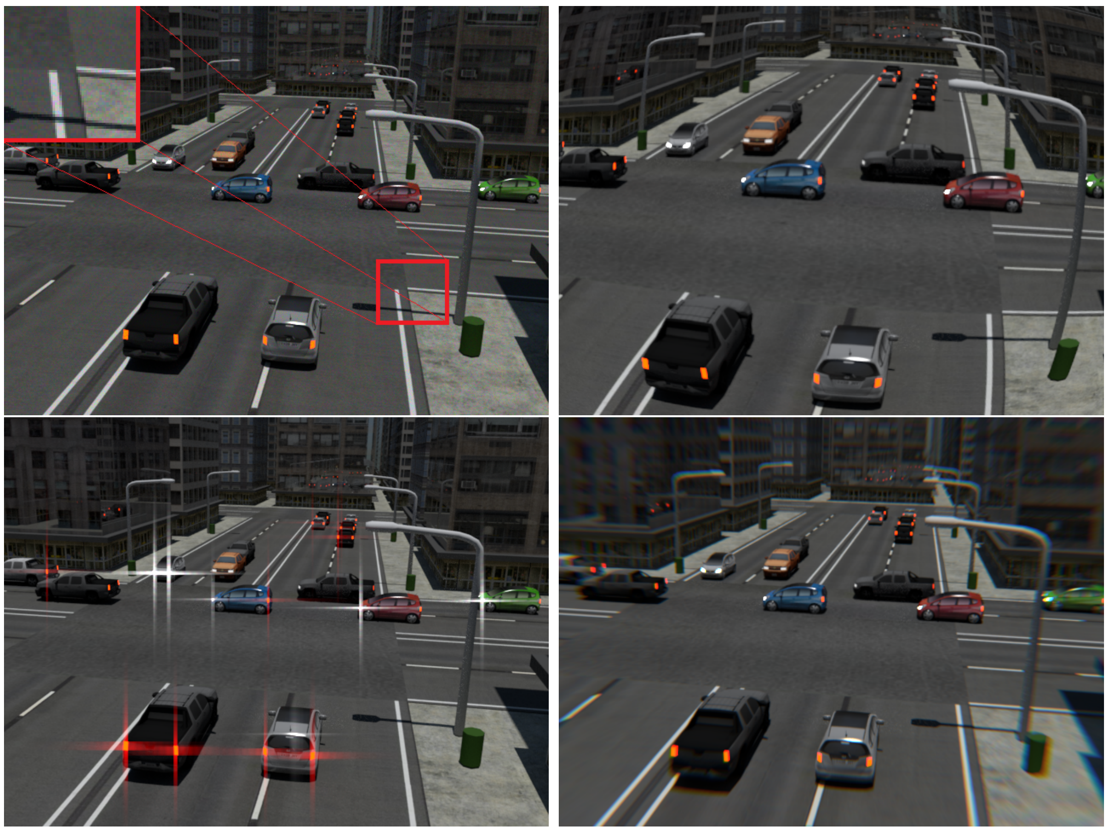


Figure 3.4: Simulation of sensor effects, *left to right*: Jitter noise (see highlighted patch), Lens distortions, camera glare, chromatic aberrations.

depending on the level-of-details incorporated into rendering. In general, different rendering methods are better suited for different applications. For example, physics-grounded rendering methods such as Monte Carlo path tracing are used for scientific visualizations and movies, whereas real-time rendering methods such as some versions of Ray tracing are used in video games.

3.1.3.1 Real time rendering

Rendering for interactive media, such as games, is calculated and displayed in real time, at rates of approximately 20 to 120 frames per second. In real-time rendering, the goal is to show as much information as possible as the human eye can process in a fraction of a second. The primary goal is to achieve an as high as possible degree of photorealism at an acceptable minimum rendering speed (usually 24 frames per second, as that, is the minimum the human eye needs to see to successfully create the illusion of movement). In fact, exploitations can be applied in the way the eye 'perceives' the world, and as a result that the final image presented is not necessarily that of the real-world, but one close enough for the human eye to tolerate. Rendering software may simulate such visual effects as lens flares, depth of field or motion blur. These are attempts to simulate visual phenomena resulting from the optical characteristics of cameras and of the human eye. These effects can lend an element of realism to a scene, even if the effect is merely a simulated artifact of a camera. This is the basic method employed in games, interactive worlds, and VRML. The rapid increase in computer processing power has allowed a progressively higher degree of realism even for real-time rendering, including techniques such as high dynamic range (HDR) rendering.

3.1.3.2 Physics-driven rendering

Animations for non-interactive media, such as feature films and video, are rendered much more slowly. Rendering times for individual frames may vary from a few seconds to several hours for complex scenes. When the goal is physics-realism, techniques such as Monte-Carlo path tracing etc. are employed. This is the basic method employed in digital media and artistic works. Techniques have been developed for the purpose of simulating other naturally-occurring effects, such as the interaction of light with various forms of matter. Examples of such techniques include particle systems (which can simulate rain, smoke, or fire), volumetric sampling (to simulate fog, dust and other spatial atmospheric effects), caustics (to simulate light focusing by uneven light-refracting surfaces, such as the light ripples seen on the bottom of a swimming pool), and subsurface scattering (to simulate light reflecting inside the volumes of solid objects such as human skin). The rendering process is computationally expensive, given the complex variety of physical processes being simulated.

For instance, see Figure 3.2. Left one is the image rendered using a ray tracer whereas the right one is the image rendered using a Monte Carlo Path Tracer of the same scene. On a normal computer, left image is rendering in real time. The right image is more physically realistic compared to the left one. However, it comes with a pay, i.e more rendering time. This trade-off depends on the application at hand, for example, video game or scientific visualizations etc. Since, one of our goals is to quantify the impact of the choice of rendering methods (appearance-drive and physics-driven methods) and their parameters on the generalization performance of the CV model trained on rendered data, we develop a tool that facilitates the use of several rendering methods in a plug-and-play manner.

3.2 Our Tool for Annotated Data Rendering

In this section, we explain the details of rendering platform we have developed for a systematic exploration of a set of scientific questions. We start by stating the requirements of an idealistic rendering platform which facilitates plug and play with different choices of rendering engines and annotations. We also describe the details of three basic rendering engines used for the experimentation provided in next chapters.

Ideally, choice of rendering process depends on a task that the experimenter intended to solve. Our aim was to develop a flexible rendering platform which meets following requirements: (i) It should provide an option of selecting a rendering kernel ranging from local to global and classical to modern rendering algorithms, (ii) It should provide a wide range of surface materials (BRDFs), volume scattering and post processing shaders, (iii) It should render images along with required annotations. Hence, we developed a rendering platform that is integrated with several rendering algorithms ranging from classical to modern methods and shaders for several image annotations such as semantic labels, depth, surface normals etc. Thanks to *Blender* [2], an open source rendering engine, met several of our requirements. We integrated the other rendering and annotations methods using its *Node* and *Scripting* interfaces.

Here, we briefly discuss the rendering methods used in our experimentation, which range from local to global illumination methods, appearance-driven, and physics driven methods. In our rendering platform, we consider three rendering engines: Lambertian shader from Direct lighting based rendering family, a Ray-tracer from the family of real-time realistic rendering (video game engines) and a Monte Carlo path tracer from the family of physically accurate methods.

3.2.1 Lambertian Shading

Lambertian shading is one of the classical example for local illumination or direct-lighting based rendering methods. It assumes all the surfaces in the scene are diffuse surfaces and models only the interaction between surface points and the light that directly come from light sources and neglect all global illumination effects. It can be expressed as a dot product between light vector and diffuse BRDF (bi-directional reflectance distribution function).

$$L_r(x) = \rho_{bd}(x, \omega_i) L_i(x, \omega_i) \cos\theta_i \tag{3.4}$$

It reflects light equally in all directions, hence, the appearance of these surfaces are view-point invariant. Many model-driven vision algorithms make an assumption that the scene surfaces are Lambertian. One image sample rendering with diffuse BRDF is shown in Figure 3.5a. These images are treated as trivial inputs to CV algorithms. Hence, the methods trained on these images are found to be more biased in reality [77]. For our work in Chapter 5, we use Lambertian shader to simulate a dataset with only direct lighting effects.

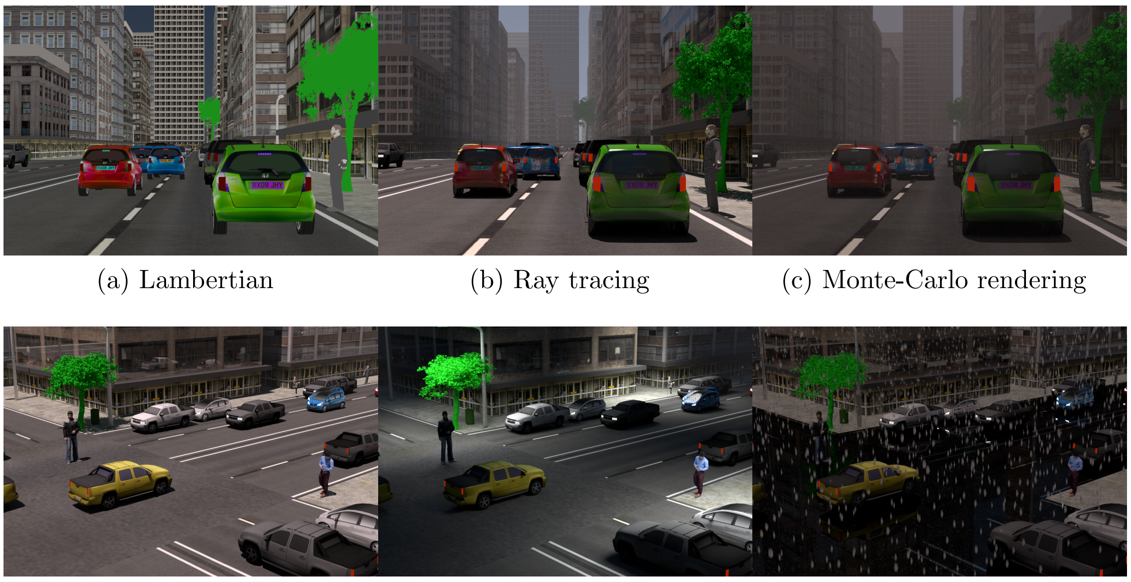
3.2.2 Ray Tracing

Ray tracing [15] can be categorized as a global illumination based rendering method that models global light interactions such as inter-reflections, scattering etc., in addition to direct lighting. However, it employs several mathematically simplifications for the sake of real time rendering. Hence it is often called as appearance driven method, as it places emphasis on the appearance of the output rather than the techniques used to derive it. These methods fire rays from the camera into the scene, and where those rays intersect the scene, they fire multiple rays to every light in the scene, and then computes the pixel value based on the material BRDF of the object with the amount of light that pixel is receiving from all the lights in the scene. This means that ray tracing can actually only compute direct lighting. All other effects, such as caustics and global illumination, are based on separate, non-physically based equations. One image sample rendering with a ray tracer is shown in Figure 3.5b. These can generate realistic images but with limited fidelity (physical accuracy). In Figure 3.5b, fog is added by post-processing fragment (pixel) buffer with the help of depth (z-buffer).

3.2.3 Monte-Carlo Path Tracing

From Equations 3.1 and 3.2, one can see that pixel value in the image is a function of nested integrals which are computationally intractable. However, some methods attempt to solve these numerically by employing Monte-Calro methods to yield approximate solutions to the integrals. These methods are used for some scientific visualizations that require very accurate and physics-grounded light computation, but at the cost of resources and time. Monte-Carlo Path Tracing (MCPT) is one such rendering algorithm, which randomly sample many quantities to compute a solution. In principle, light rays bounce around the scene (these bounces are calculated as random directional values), accumulating all the values needed to solve the rendering integrals. The final image quality of the render is determined by a parameter *Samples-Per-Pixel* (spp). The more samples per pixel, the more refined the image, thus, more accurate. In this work, we consider Monte-Carlo-Path-Tracer (MCPT) engine [82, 81] to render the images of high fidelity. Some examples rendered with *MCPT* are shown in Figure 3.5c, 3.5d, 3.5e, and 3.5f.

In summary, *MCPT* is a more physics-grounded, but computationally intensive approach to rendering. *Ray tracing* is faster and more efficient, but it uses mathematical heuristics for the effects of global illumination/caustics. In the end, there are different solutions for different applications in CG. However, our interest is to use simulated images to train vision systems. How do visual realism and physical fidelity affect the generalization of the trained model? To address this question, we resort to empirical approach (see Section 5.4) in which we render image sets with different rendering engines, compare the performances (on a real world dataset) of the models trained on these sets independently.

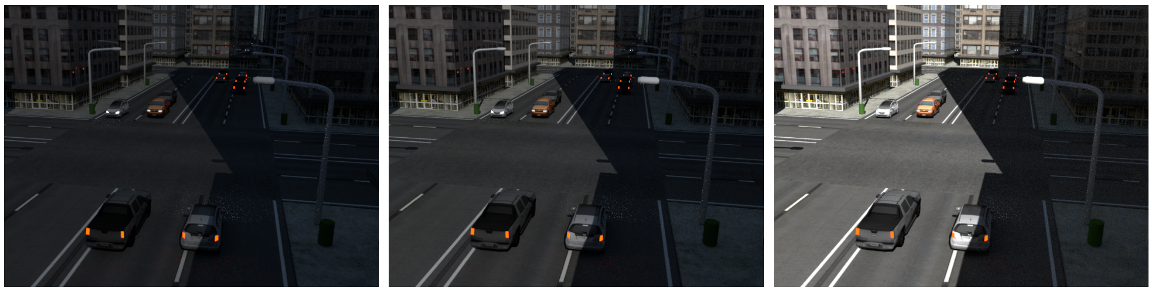

(a) Lambertian (b) Ray tracing (c) Monte-Carlo rendering

(d) Day light (e) Night (f) Rain

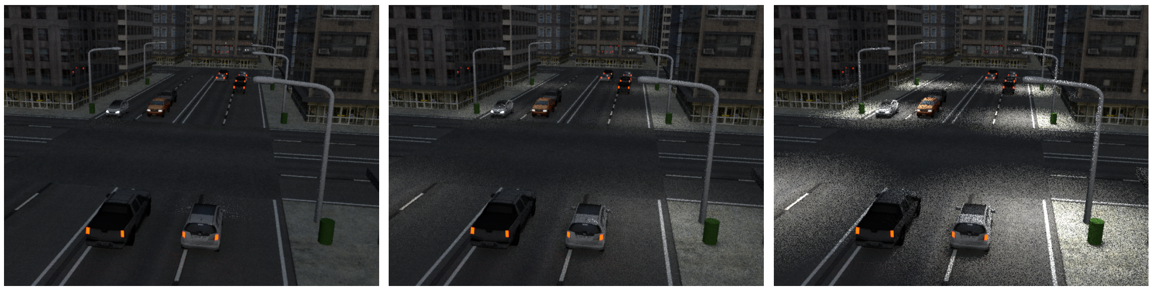
Figure 3.5: Rendering fidelity and Virtual scene diversity. This work aims to quantify the impact of photorealism and physics fidelity on transfer learning from virtual reality. (a)-(c): Images of same scene state rendered with different rendering engines. (e)-(g): Same scene under different lighting. (d) and (h) semantic labels. Color coding scheme for labels is same as [16].

3.3 Photometry

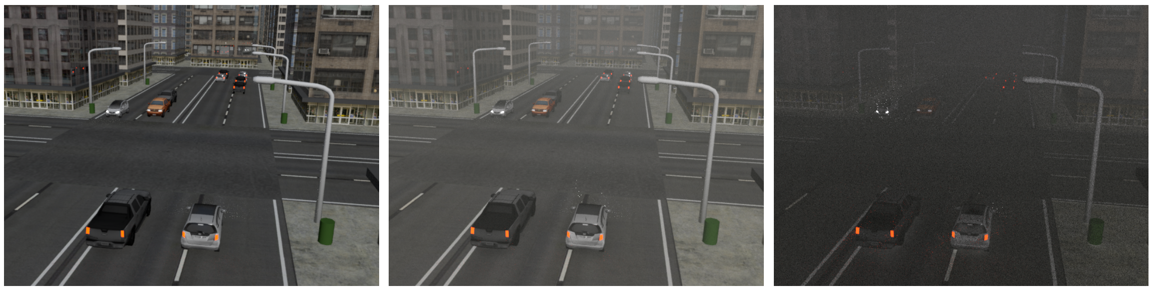
In addition to the parameters of the rendering engine, we need to select lighting and weather scattering parameters of the scene. Several light models such as sunlight, street light etc. are provided with Blender. These are parametrized by color spectrum and intensity. We also scripted some light variations such as morning-to-noon and noon-night etc. In our work, we use uniform distributions on several parameters including light source intensity, location, orientation etc on permissible ranges. We also used street lamps at fixed locations in 3D world coordinate system to create night lighting. The intensities of these lamps are conditioned on the sunlight. When we use ray tracer for rendering, weather effects are added with post-processing shaders. For *MCPT* rendering, physics-inspired shaders for volume scattering effects are used to create weather effects such as fog, haze, and mist etc. These are parametrized by



(a) Global light intensity variations



(b) Night light variations



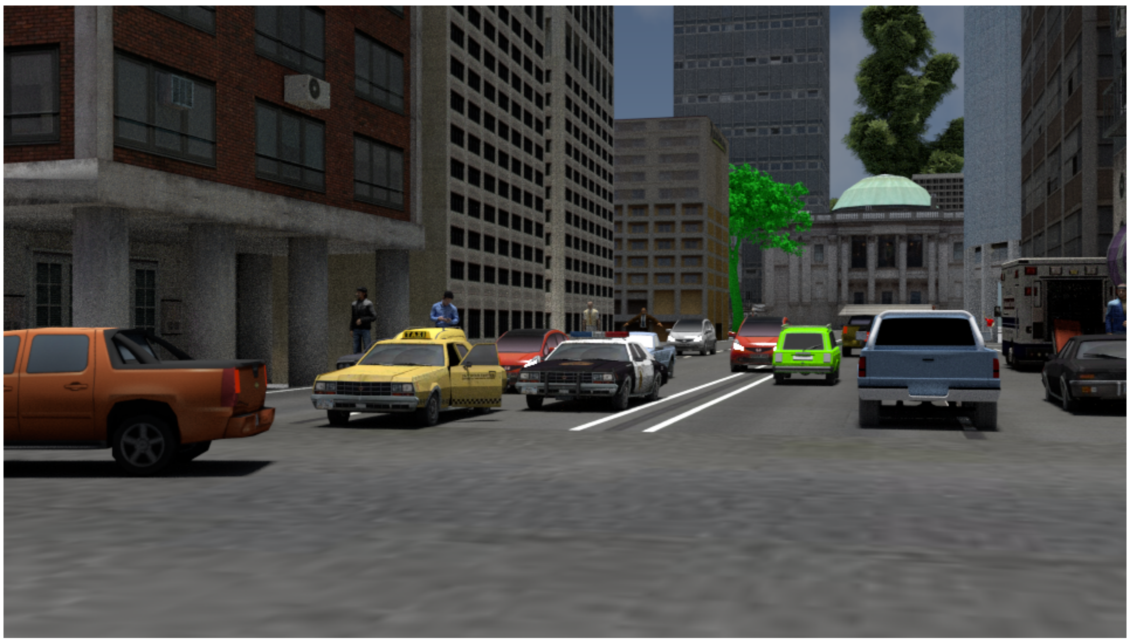
(c) Fog density variations

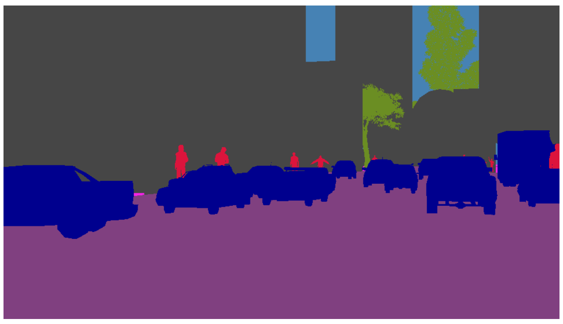
Figure 3.6: Simulated samples

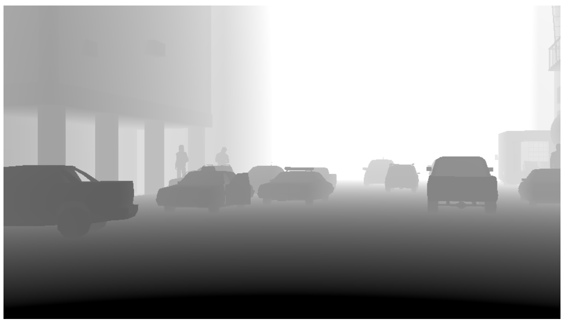
particle size, density, and anisotropy. These are proven to be good approximations for theoretical functions and well suited for Monte Carlo rendering methods. Dynamic weather effects such as rain and snow can be simulated with help of particle systems [62] with water droplets or snowflakes as particles. The image rendered under different lighting and weather settings are shown in Figure 3.5d, 3.5e, and 3.5f. In the following experimentation, we did not consider dynamic weather effects such as rain and snow etc.

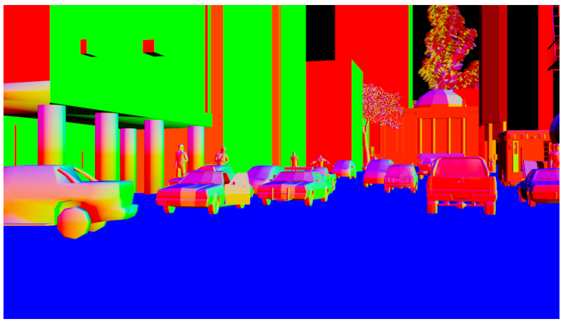
3.4 Annotations

Nature of required annotations depends on the intended goal of vision systems, which varies from overall System evaluation (e.g. Performance assessment, offline learning of algorithm, etc.), to component evaluations and Model learning (ideal signals and perturbations). Annotations can be bounding boxes around objects, specifying groups, identifying edges, region labels, etc. Annotations in systems engineering context are essentially used for contextual model learning. Hence, we implement a multitude of procedural shaders which are responsible for rendering multiple image modalities or groundtruth (from local level to semantic level representations) including *depth, surface normals, reflection, shading, diffuse, specular, direct light components, optical flow, geometric flow, trajectories, semantic labels, bounding boxes, shadows* etc. A sample of these modalities is provided in Figure 3.7.


(a) RGB image


(b) Semantic class labels


(c) Depth


(d) Surface normals

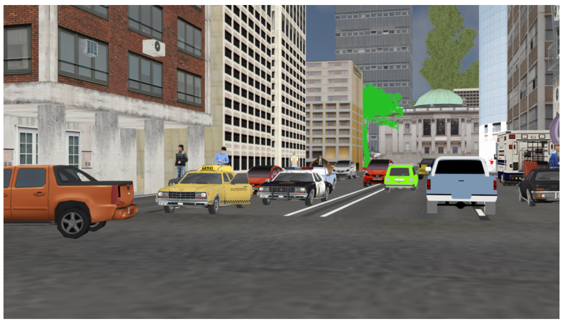

(e) Diffuse reflections

Figure 3.7: An image sample and corresponding pixel-level annotations

4 Stochastic Scene Generation

Although CG can simulate very realistic images/videos, generating open world scenes in large scale is not easy. Automatic scene generation is essential as the manual creation of 3D scene states requires a lot of man-hours. As mentioned already, Systems engineering perspective also proposes to design a mathematical model which dictates input states to CV system. We aim to engineer a probabilistic scene model which facilitates stochastic generation of diverse 3D scene states. Automatic or semi-automatic content generation is also one of the interests of computer graphics designers. (Semi) automatic large content creation with procedural grammars has been attempted in [53] and a recent survey of these methods can be found in [71]. However, they might produce highly correlated scenes for two runs of the systems as scene generation is based on given axioms and deterministic logic.

Probabilistic models are a powerful way to describe the subjective priors and constraints on scene layout of man-made and natural objects. For example, the spatial arrangement of objects (for examples, buildings, and trees) in a city can be described by a set of probabilistic constraints. Though several probabilistic scene models exist, most of them are not suitable for practical usage as they require complex inference/sampling methods. Hence, we design a simple but effective scene model that leverages the existing 3D CAD object shapes to create large scale 3D layouts.

A good scene generative model is supposed to cover a wide variety of scene conditions in terms of geometry, lighting and object's movement. Hence, the three major aspects of the scene model are: Geometry, Photometry, and Dynamics. In this work, we skip the modeling of scene dynamics as we work with static scenes and images. In this chapter, we restrict our goal to model urban traffic scene environments.

4.1 Scene Geometry

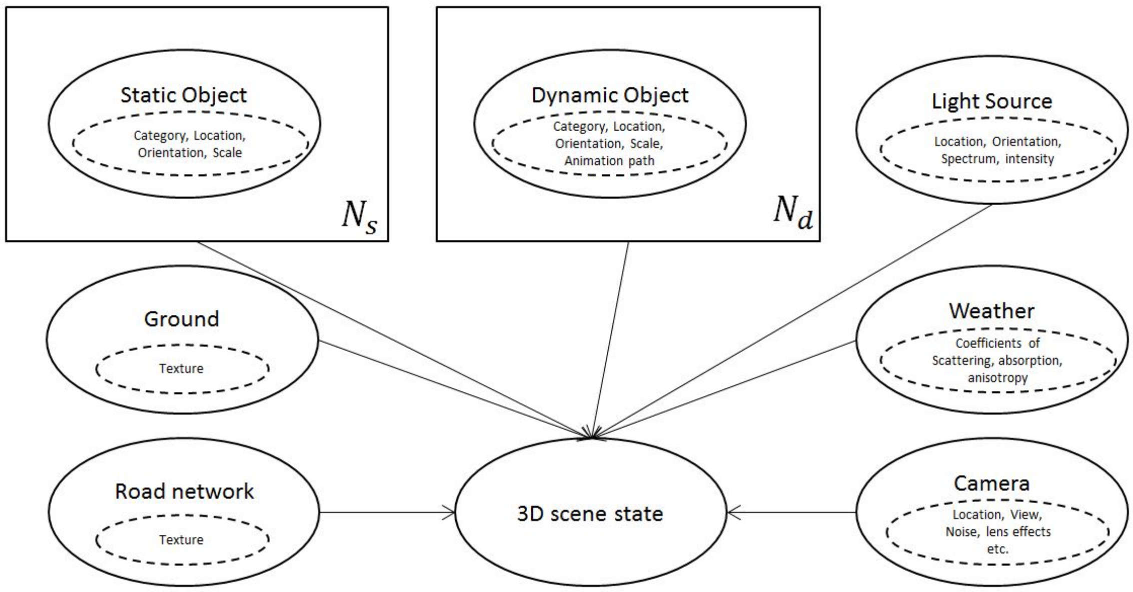
In the context of urban scenes, we model a scene geometric instance as a sample from a probability distribution of a spatial stochastic process with additional attributes. Modeling scene geometry and object shape is getting easier due to the availability of large-scale 3D CAD shape repositories such as Google's 3D warehouses etc. We collected a variety of CAD shapes and textures from the web, and place them in a world's coordinate system according to the realizations of *Marked Point Processes* (*MPP*).

Marked point processes are frequently being used in numerous fields such as such as astronomy, biology, ecology, geology, physics, economics, and telecommunications etc. These have been introduced to computer vision fields by the work [4]. Later, these got extended further by many researchers, for instance, [43, 76]. These stochastic processes can be considered as a generalization to conventional Markov random fields where random variable associated not with pixel values but with geometric shapes describing a bounded region. In the following, we introduce some concepts of marked point processes.

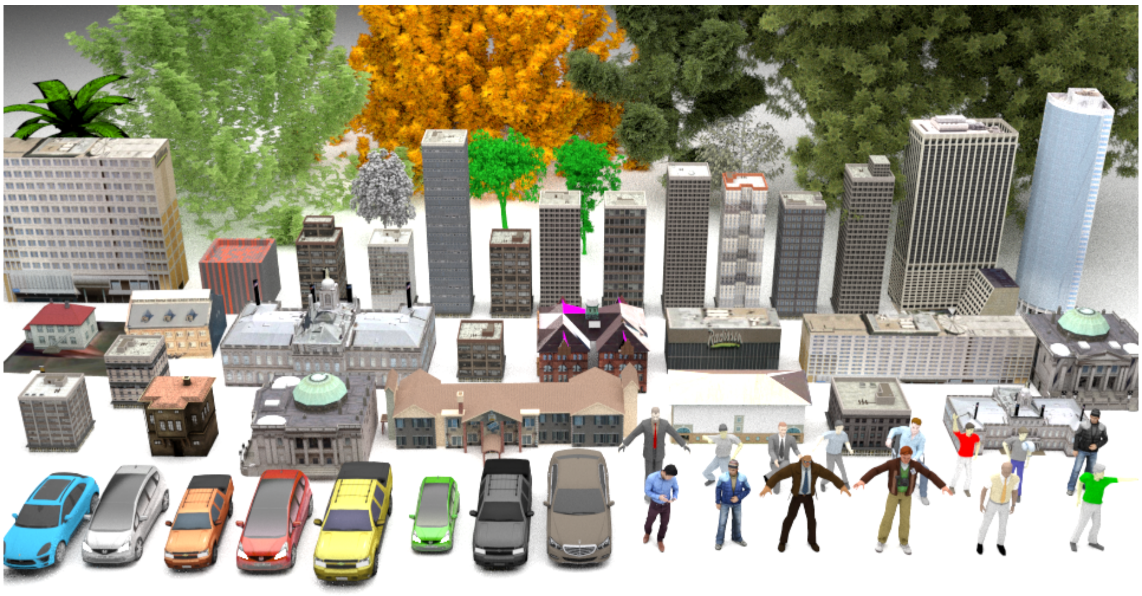
4.1.1 Marked Point Processes for Scene Geometry

Prior knowledge about spatial/temporal distributions of a set of objects can be modeled with stochastic processes such as point processes [36]. A realization of the point process consists of a random countable set of points $\{o_1, ..., o_n\}$ in a bounded region $\mathbf{O} \in R^d$. A marked point process couples a spatial point process Z with a second process defined over a *mark* space M such that some random marks or attributes $m_o \in M$ are associated with each point $o \in Z$. For example, a 3D marked point process of cuboid marks has elements of the form $o_i = (p_i, [c_i, l_i, b_i, h_i, \psi_i])$ specifying the location (p), object class (c), size dimensions (l,b,h) and orientation (ψ) of a specific bounding box (cuboid) in the scene. A sample of realization from MPP can be seen in Figure 4.2a.

In this work, we use MPP to incorporate our prior knowledge about the spatial patterns in city geometries (extrinsic size and transformation of objects such as buildings and vehicles etc). Thus, the realization of MPP in this work consists of an object location p defined on a bounded subset of R^2 (xz ground-plane), together with a mark m defining type and a 3D CAD shape to import and place at a point p. In other words, we model the objects in the scene as a set of configurations from an MPP that incorporates prior knowledge such as expected sizes of buildings, trees, people and vehicles on the scene of knowledge about the scene regions where these objects will not appear. We denote the prior term for an object as $\pi(o_i)$, and assume independence among the objects. The mark process is assumed as independent from the spatial point process, so that priors in MPP would be factored as: $\pi(o_i) = \pi(p_i)\pi(m_i)$. However, this common approach ignores obvious and strong correlations between the size and orientation of objects. Hence, a conditional


Static Object
Category, Location, Orientation, Scale
N_s
Dynamic Object
Category, Location, Orientation, Scale, Animation path
N_d
Light Source
Location, Orientation, Spectrum, intensity
Ground
Texture
Weather
Coefficients of Scattering, absorption, anisotropy
Road network
Texture
3D scene state
Camera
Location, View, Noise, lens effects etc.

(a) Semantic diagram for our virtual world model: rectangle boxes represent MPP, N_s and N_d denote the number of static and dynamic objects respectively in the virtual world



(b) A few samples from our CAD repository

Figure 4.1: Scene Generative Models

mark process is introduced for cuboids representing shape and orientations of a 3D bounding box, conditioned on the location and shape, leading to a factored prior of the form:

$$\pi(o_i) = \pi(l_i, b_i, h_i, \psi_i | c_i, p_i)\pi(c_i)\pi(p_i) \tag{4.1}$$

The prior for object class (type such as building, tree, pedestrian and vehicle classes etc) distributions are modeled with uniform distributions. This means object type follows a uniform distribution, and given object type, the shape dimensions (3D bounding box) are *i.i.d.* One can also bootstrap the scene models by learning the parameters through training using real-world data.

Conditional mark process: We represent priors for $\pi(l_i, b_i, h_i, \psi_i | p_i, c_i)$ as simple parametric distributions (such as uniform distributions) on the bounded regions. For example, the heights of pedestrian and vehicle are modeled with uniform distributions with 1.6 ± 0.3 and 1.7 ± 0.3 meters respectively. ψ_i (orientation with which object should be placed on the ground or road) is allowed to vary from -180 to 180 degrees with uniform distribution.

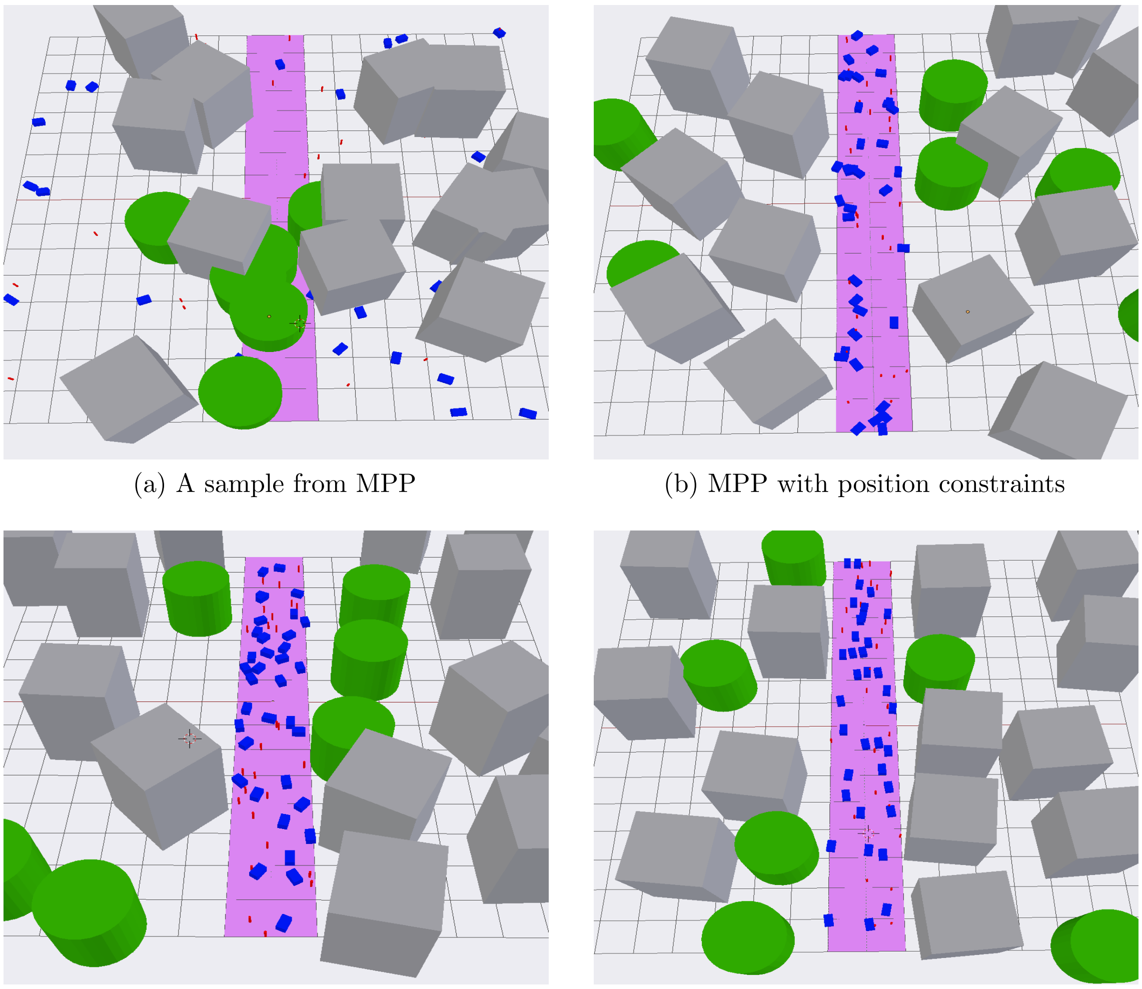
3D CAD shapes and Textures as Marks. The object shapes (for given object identity) are randomly selected from the 3D CAD repositories and resized according to the sampled (l, b, h). We have collected a rich set of 3D models for each object category (buildings, grounds, pedestrians, and vehicles etc) from the web[1]. Some of the 3D CAD shapes are shown in Figure 4.1b. These 3D models are indexed according to their object category. However, parametric 3D object shape can be modelled with the distributions [30] such as Boltzmann machines [83] or Bernoulli mixture distributions [26].

4.1.2 Scene Layout constraints as Factor potentials

One can simply assume the independence in between marks or attributes of the objects for simplicity in sampling from the world models. Scene instances generated using the statistical independence assumption between marks will contain spatial overlaps that are physically improbable as shown in Figure 4.2a. Hence, some inter-dependencies between marks (such as spatial nonoverlap, co-occurrence, and coherence among the instances of object classes etc.) is incorporated with the help of Factor potentials.

Factor potentials have been traditionally used to incorporate a set of constraints into modeling. The work [87] has employed factor potentials to encode the constraints furniture arrangement, however, their system requires user interaction. Inspired from

[1]3dwarehouse.sketchup.com, 3dmodelfree.com, quality3dmodels.net, tf3dm.com


(a) A sample from MPP

(b) MPP with position constraints

(c) MPP with nonoverlap constraint

(d) MPP with all constraints

Figure 4.2: Marked Point Processes with Factor Potentials

these works[43, 73], we formulate scene generation as a constrained sampling problem. Hence, probabilistic distributions over scene layouts from MPP is formulated using Gibbs equation:

$$\pi(o) = \frac{e^{-E(o)}}{\int_O e^{-E(o)}} \tag{4.2}$$

where $E(o)$ introduces prior knowledge on the object layouts by taking into account factor potentials between the objects. In our current context (i.e. urban scenes), these factors could be corresponding to a set of layout constraints such as support relationships (F_p), spatial exclusion between the bounding boxes of objects (F_b), and alignment to road (F_a). Then the energy of a realization of MPP is expressed as,

$$E(o) = \sum_{o_i, o_j \in O} w_p F_p(o_i) + w_b F_b(o_i, o_j) + w_a F_a(o_i, r) \tag{4.3}$$

where w_p, w_b, and w_p are weights associated with the factors F_p, F_b, and F_p respectively. o_i and o_j are two different objects while r represents road in MPP. These factor potentials are explained below.

4.1.2.1 Position Constraint

A valid scene layout must obey some very basic criteria of feasibility observed in real world scenes. First such criterion we consider here is a constraint of the position of objects. Placement of an object in the scene layout depends on its category. For example, buildings and trees should be placed on the ground; and vehicles and pedestrians should be placed on the road. Any deviation from this constraint is penalized by the factor:

$$F_p(o_i) = \begin{cases} max(0, b_i - d_{ir}) & c_i \in \{building, tree\} \\ min(0, b_i - d_{ir}) & c_i \in \{vehicle, pedestrian\} \end{cases} \tag{4.4}$$

where b_i is breadth of the bounding box of the object o_i and d_{ir} is euclidean distance between object's position (p_i) and a nearest road point. A sample of MPP process with this constraint is as shown in Figure 4.2b. Please note that we use a fixed ground plane and Manhattan-like road network, although their texture maps are randomly selected.

4.1.2.2 Non-overlap Constraint

Bounding boxes of any two objects must not intersect with each other. The factor we use here has the following representation:

$$F_b(o_i, o_j) = max(0, b_{ij} - d_{ij}) \tag{4.5}$$

where b_{ij} is the sum of the breadths of the bounding boxes of two objects (o_i and o_j). d_{ij} is the euclidean distance between the positions of two objects. Generally, d_{ij} is greater than or equal to b_{ij} is they are not intersecting each other. For such cases, penalizing factor would be zero or some positive value otherwise. Please refer Figure 4.2c to see the effect of this factor on the scene layout.

4.1.2.3 Alignment with Road

For scenes to be visually pleasing, orientations of objects (for example building and vehicles) should be likely to be aligned with the road. Hence, we add another factor to increase the likelihood of such object states. It has the following representation:

$$F_a(o_i) = (\psi_{ir} - \psi'_{ir})^{\alpha} \tag{4.6}$$

where ψ_{ir} is the difference between orientations (ψ) between object and nearest road segment. ψ'_{ir} denotes the trade-off that one wants to allow. We fix $\alpha = 2$ which represents quadratic penalizer. Please see the effect of this factor in Figure 4.2d.

4.1.3 Limitations

Our model uses a fixed template for Manhattan-like road network and ground planes, though the textures are loaded randomly. Hence, it may not be able to capture variations like curvy roads, complex junctions and terrain situations.

4.2 Scene Photometry

Photometry of Scene state is one of the major factors which influences the visual appearance of the image. It includes modeling the light sources, weather, and camera. However, graphics tools readily provide a set approximated models for light sources, volume scatterers and camera models. Several light models such as sunlight, street light etc. are provided with Blender. These are parametrized by color spectrum and intensity. In our work, we use uniform distributions on several parameters including light source intensity, location, orientation etc on permissible ranges. We also used street lamps at fixed locations in 3D world coordinate system to create night lighting. The intensities of these lamps are conditioned on the sunlight. When we use ray tracer for rendering, weather effects are added with post-processing shaders. For *MCPT* rendering, physics-inspired shaders for volume scattering effects are used to create weather effects such as fog, haze, and mist etc. These are parametrized by particle size, density, and anisotropy. These are proven to be good approximations for theoretical functions and well suited for Monte Carlo rendering methods. Dynamic weather effects such as rain and snow can be simulated with help of particle systems [62] with water droplets or snowflakes as particles. The image rendered under different lighting and weather settings are shown in Figure 3.5d, 3.5e, and 3.5f. In the experiments provided in this thesis, we did not consider dynamic weather effects such as rain and snow etc.

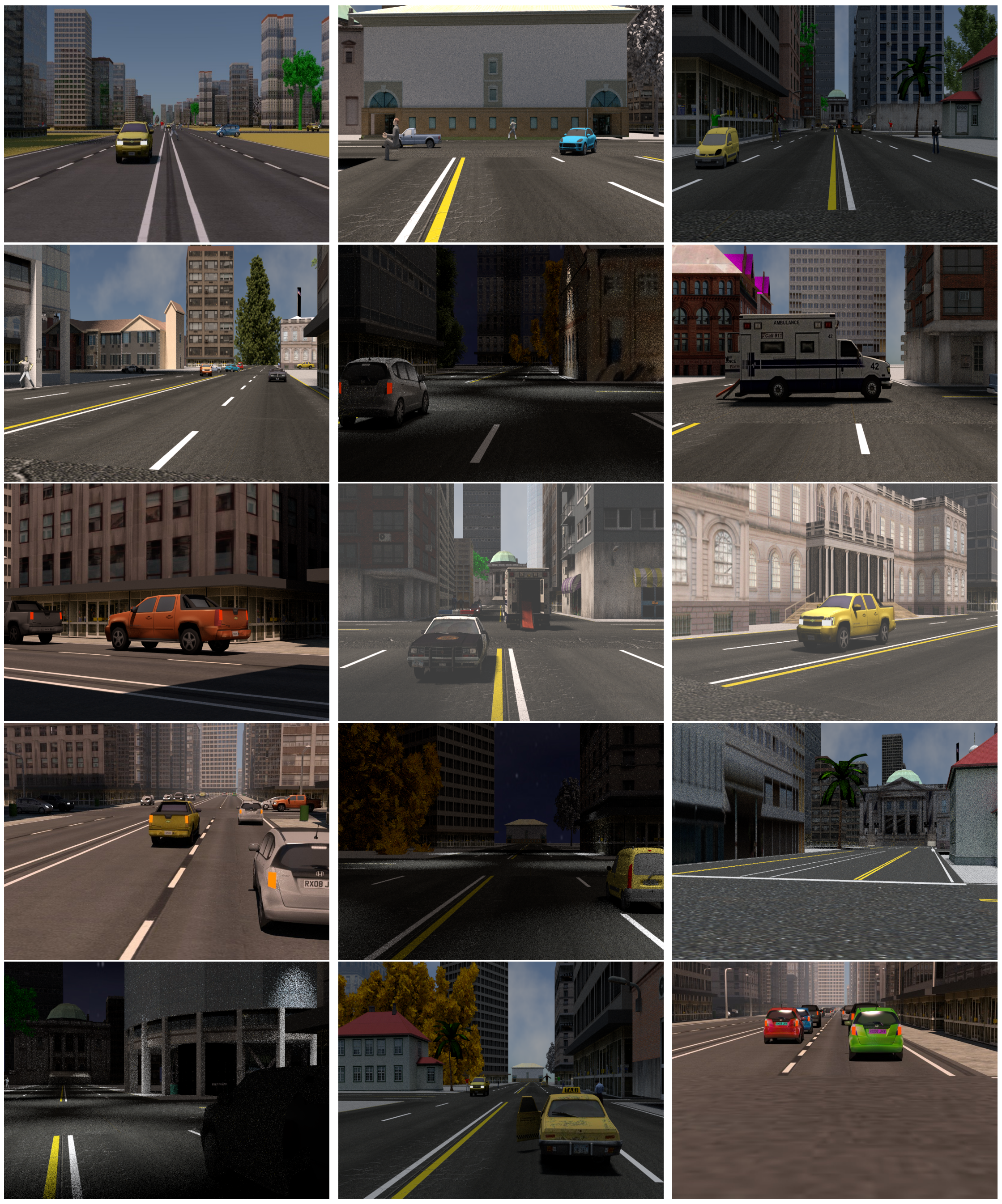


Figure 4.3: A few samples from our scene generative model

5 Impact of Rendering Fidelity

In this chapter and the following ones, we utilize our data simulation tools to address the some fundamental scientific questions about the use of synthetic imagery to train modern CV systems, especially deep convolutional neual networks (DCNN). This chapter particularly quantifies the impact of different rendering factors on the generalization performance of CV systems on the real-world data when it is trained using synthetic data generated from our tools. Since pixel-level annotations such as semantic labels are one class of the groundtruth informantion which is time consuming to acquire on the real-world, we consider pixel-level semantic labeling as a task of interest. With the advent of fully convolutional networks (FCN), several DCNN based architectures have been proposed for semantic labeling, among with DeepLab [13] is one of the state-of-the-art architectutes. Hence this work considers a case study involving pixel-level semantic labeling in traffic scene context with DeepLab architecture.

5.1 Semantic Labeling

Pixel-level semantic labeling is considered as a low-level vision task that can play a central role in holistic scene understanding. Before the resurrection of deep neural networks into CV fields (roughly around 2010), random fields with hand-engineered features had been a major paradigm to design algorithms for pixel level labeling. Most labeling methods use conditional random fields (CRF) [72] with pair-wise factor potentials that encode the label smoothness constraints in a neighborbood. This neighborbood in general is defined over a local grid in a graph stucture, i.e., label of each pixel is connected to its immediate neighbors. An important work to mention in this class of algorithms is TextonBoost [69] which uses texture-layout filters (hand engineered features based on textons [38]). The recent developments on CRF devised efficient inference algorithms for fully connected CRF graphs in which each pixel's label is connected every other pixel's label node. This has been demonstrated to improve the performance of TextonBoost by around 4% on MSRC-21 dataset [69]. This class of algorithms may not require large amounts of labelled sets to tune their parameters. Hence, MSRC-21 set that contains 94 images with groundtruth labels for

21 object classes. It has been reported that labeling a single image took 30 minutes on average.

With the resurrection of deep neural networks into CV fields, Convolutional neural networks (CNN) have become a standard choice for almost all CV applications. Image classification networks such as VGG-16 [70] has been extended as FCN [46] to work for pixel level classification tasks such as semantic labeling. These FCN architectures employ some stategies to upsample the feature maps after some layers in order to get an output of similar resolution on the other end of the architectures. Long et al [46] uses an upsampling scheme is based on deconvolutions and mixing information across layers using skip connections. However, the work of Chen et al [14], known as DeepLab, highlights standard convoltion but with upsampled filters (known as Atros convoltution in Wavelet community) as a powerful tool for dense prediction applications. This type of convolutions explicitly control the feature resolutions within final layers of DCNNs without increasing the number of parameters or the amount of computation. It also suggests to use a fully connected CRF inference as a post processing step to increase the localization accuracy of labels. DeepLab architecture outperforms FCN by 1.8% on PASCAL-context dataset (benchmark for generic semantic segmentation). Hence we select DeepLab as system to be trained and evaluate virtual-reality-based-training for semantic segmentation task in traffic scene context.

Large amounts of labeled data might be required to train these DCNN based architectures. However as mentioned already, the acquisition of real-world data with pixel-level annotations is costly in terms of money and efforts, factors that are slowing down the collection of new large scale datasets such as ImageNet [18]. Especially for traffic scenes or autonomous driving context, diversity of image samples and quality of annotations in the existing datasets are much lower. For instance, CamVid [7] dataset consists of only 701 images with pixel-level labels for 11 object classes. Similarly, the very popular KITTI benchmark suite [27] for autonomous driving provides only 430 labeled images for semantic segmentation. Recently, Cordts et al [16] collected a large scale dataset, CityScapes, with 3475 images with finer pixel-level semantic annotations. This is recorded on the streets of several European cities in Germany, Switzerland and France in different seasons. However, the bias introduced by the collection processes in a specific city or cities is also a common limitation. Recently, a DCNN model trained on CityScapes train subset is shown to be performing poorly on a testing dataset from American cities while showing excellent performance on CityScapes test subset. Hence, preparing a high quality of dataset would require capturing images under a wide variety of seasons in different countries and traffic conditions. Such data collection processes are practically infeasible due to several factors such as money and efforts. For these reasons, a promising alternative could be to utilize simulation tools that synthesize urban scenes in a vast variety of conditions to simulate the training image sets along with required annotations.

However, the CV models that are naively trained on simulated dataset might suffer with severe dataset-bias problems. Approximations made in virtual world models and rendering processes influence the statistics of rendered outputs, and thus, biases the learned classifier. For better usage of the synthetic training sets, we believe that the impact of simulation factors on the system's generalization has to be studied systematically. The type of rendering algorithm and its parameters are one of the factors that majorly influence the appearance (photorealism) of image and its statistics. Hence, we first aim to quantify the impact of rendering engine on the generalization performance the DeepLab model trained simulated data.

Towards this end, we conduct experiments to quantify the following:

- Effects of Photorealism: Here we compare results obtained from training data using Lambertian shaders against photorealistic methods such as Ray-tracing and Monte-Carlo Path tracing.

- Physics Fidelity: Here we compare results obtained by training with Ray-tracing vs Monte-Carlo Path tracing (MCPT). MCPT datasets are physically more realistic than Ray-traced ones, though they both look realistic to human-eye.

- The impact of Computational Approximations: Here we evaluate the variation in performance as a function of number of Monte-Carlo samples (samples-per-pixel, spp) in MCPT.

5.2 Data preparation

5.2.1 Simulated Sets

We simulate 5000 images sampled from the scene generative model (described in Chapter 4) and rendered with different rendering settings, along with pixel-wise object labels (the classes include: vehicle, pedestrian, building, vegetation, road, ground, and sky). To measure the impact of choice of rendering method and its computational-approximation, we render these images Lambertian shader, ray-tracer, and MCPT with increasing computational-approximation (samples-per-pixel, spp) ranging 10 to 130 with a step size of 30 ($spp = 10 : 131 : 30$ *python notation*). It results in seven image datasets of same scene states. We use the terms *"Lambertian"*, *"RayTrace"*, and *"mcpt_X"* to denote the sets rendered with Lambertian shader, ray tracer and path tracer respectively, where X being number of *spp* used in rendering.

5.2.2 Real-world datasets

For comparison purposes, we use a real-world dataset, CityScapes[16] which is recorded on the streets of several European cities. It provides a diverse set of videos with a public access to 3475 images (train-val) that has finer pixel-level annotations for semantic labels. We divide the database into two disjoint subsets for training, validation (3000 images, named as *"CS_train"*) and testing (475 images, named as *"CS_val"*) purposes. This dataset was adapted to the 7 class labels mentioned above.

5.3 Deep Network architecture

As mentioned already, we select a state-of-the-art DCNN-based architecture, i.e. DeepLab [13] as a baseline in our experimentation. Thanks to the authors of DeepLab for making their implementation publicly accessible.

5.3.1 DeepLab

DeepLab [13] is a modified version of VGG-net [70] to operate at original image resolutions, by making following changes: (a) replace the fully connected layers with convolutional ones, (b) skip the last subsampling steps and upsample the feature-maps by using *Atros* convolutions. It still results in coarser map with a stride of 8 pixels. Hence, targets (semantic labels) during training are the ground truth labels subsampled by 8. We also add batch normalization layers after every convolutional layer as it might reduce covariate shift across minibatches. During testing, bi-linear interpolation followed by fully connected conditional random field (CRF) was used to get final label maps. To harmonize the final layer's output with our data annotations, we modify the last layer of DeepLab from 21-class to 7-class (including: vehicle, pedestrian, building, vegetation, road, ground, and sky).

5.3.2 Training

Our DeepLab models are initialized with ImageNet pre-trained weights to skip longer training times. Stochastic gradient descent method and cross-entropy loss function are used with an initial learning rate of 0.001, a momentum of 0.9 and a weight decay of 0.0005. We use mini-batch of 4 images and learning rate is multiplied by 0.1 after every 2000 iterations. High-resolution input images are down-sampled by a factor 4. Training data is augmented by horizontal flipping and random crops from the original resolution images etc., which yields four times the data. As stopping criteria, we use a fixed number of SGD iterations (50,000) in all our experiments. In the CRF postprocessing, we use fixed parameters in the CRF inference process (10 mean field iterations with Gaussian edge potentials as described in the work [13]) in all reported experiments. The CRF parameters are optimized on a subset of 300 images, randomly selected from the dataset.

5.3.3 Performance Measure

Mean Intersection-Over-Union (mIoU) is the standard performance measure for segmentation purposes due to its representativeness and simplicity. It computes a ratio between the intersection and the union of two sets, in our case the ground truth and our predicted segmentation. mIoU ratio is computed as the number of true positives (intersection) over the sum of true positives, false negatives, and false positives (union).

$$mIoU = \frac{1}{k+1}\sum_{i=0}^{k}\frac{p_{ii}}{\sum_{j=0}^{k}p_{ij}+\sum_{j=0}^{k}p_{ij}-p_{ii}} \tag{5.1}$$

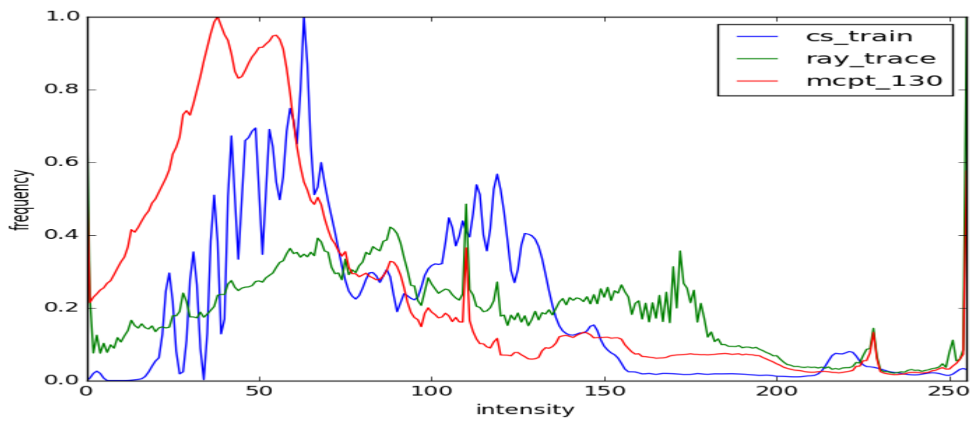
Here $k+1$ denotes number classes inclusing void or background class and p_{ij} is the amount of pixels of class i inferred to belong to class j.

5.4 Experiments and Results

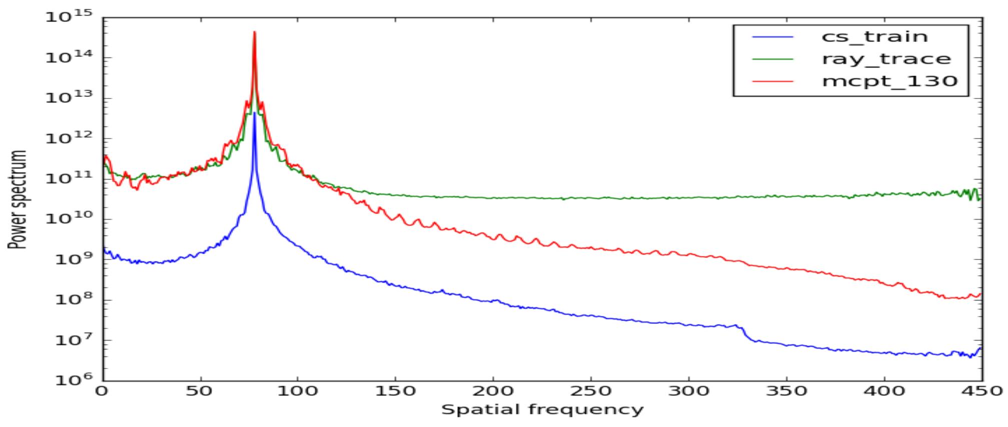
5.4.1 Input Image statistics

We start by comparing the image statistics of datasets to know how rendering method influences the pixel statistics in image space. Figure 5.1a and 5.1b display the histograms (normalized) and power spectra computed over all images of the sets. From the figure, the pixel intensity distributions of virtual and real worlds are quite varied, however, their modes seem to be closer on intensity axis. Power spectra of the simulated datasets (both *RayTrace* and *mcpt*_130) deviate from the real world spectrum more at higher frequencies. The reason for this might be fact that higher frequency contents such as shadow and object boundaries are sharper in simulated images compared real world images.

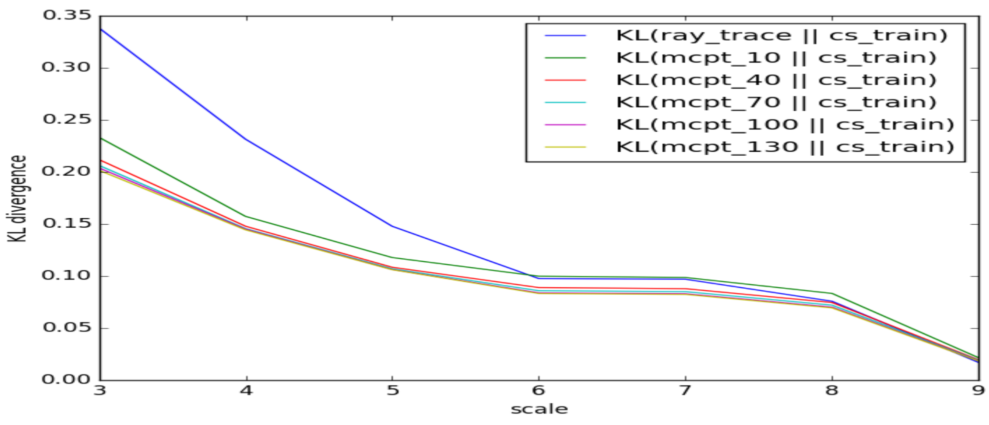
We also compute the Gabor responses of each set with Gabor filters of 31x31 size, but varying scale (σ) parameter (*scale* = 3 : 9 : 1 *python notation*). In Figure 5.1c, we plot KL divergence of distributions of Gabor filter responses between simulated



(a) histograms



(b) power spectra



(c) KL Divergence of distributions of Gabor responses as a function of scale of Gabor filter

Figure 5.1: Image statistics across datasets simulated with different rendering settings: Simulations appear to be more deviated at higher frequencies and lower scales.

Table 5.1: Comparing the performance of DeepLab when trained with simulated datasets of different computational-approximation and rendering methods. Figures in last column denote the time required for rendering an image with corresponding engine and parameter settings.

Training dataset	Validation dataset	mean IoU	vehicle	pedestrian	building	vegetation	road	ground	sky	time (in sec)
Lambertian	*CS_val*	24.46 ± 9.69	13.19	15.41	17.58	20.99	21.2	29.35	53.52	0.001
RayTrace	*CS_val*	44.81 ± 4.15	46.36	37.72	44.13	49.88	42.58	40.27	52.75	20
mcpt_10	*CS_val*	34.10 ± 6.44	30.19	28.82	31.12	30.22	32.48	29.26	56.65	5
mcpt_40	*CS_val*	50.00 ± 5.88	48.49	53.39	63.34	50.78	46.66	34.27	53.09	34
mcpt_70	*CS_val*	49.82 ± 6.04	43.57	42.17	65.26	52.73	49.85	42.57	52.59	67
mcpt_100	*CS_val*	53.82 ± 6.14	55.8	56.45	62.36	60.83	51.77	34.37	55.21	313
mcpt_130	*CS_val*	52.15 ± 6.58	56.61	47.26	54.98	63.08	56.98	36.31	49.86	547
Variance		±5.56	±5.08	±5.10	±3.25	±5.1	±3.06	±2.85	±1.46	NA

datasets and real-world *CS_train* dataset against filter scale. For all plots, the divergence of corresponding simulated set from reality seems to be decreasing with the scale. That means simulations are more deviated in local statistics than global averages. Moreover, the difference between *RayTrace* and *mcpt* sets is also decreasing with increasing scales. How do these deviations in input statistics propagate through the training stages of DeepLab?

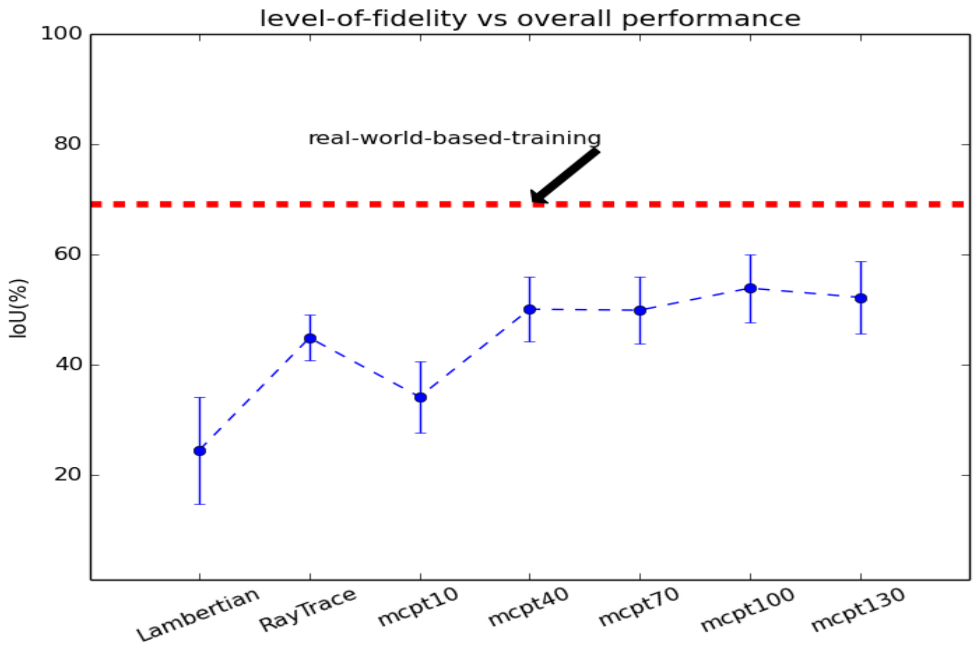
5.4.2 Performance of Virtual-world-based-training

In order to evaluate the impact of photorealism and fidelity on the performance of the model trained on virtual-world data, we train the DeepLab independently on our simulated datasets. We then compare the performances of these trained models on a reference real world dataset, i.e. CityScapes validation set (CS_val).

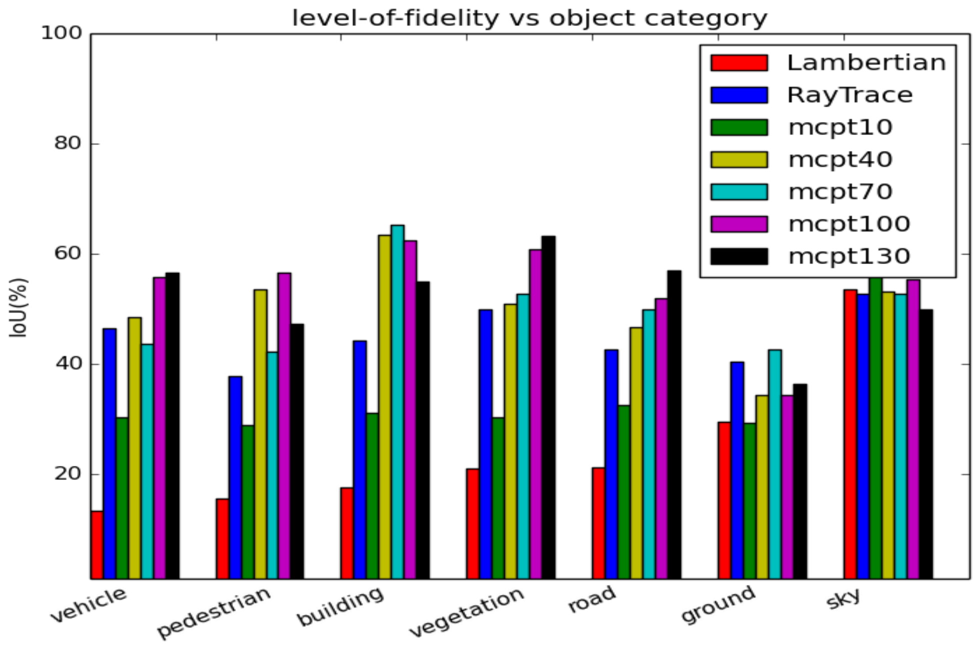
As shown in Figure 5.2a, we plot the corresponding $mIoU$ measures to show how the performance of trained model on CS_val varies with levels of realism and fidelity in simulated training data. In all configurations, training with simulations seems worse compared to real-world-based-training (red dotted line in the figure). This bias could be due to sampling diversity (and its lack of closeness to reality) in the virtual world models and computational-approximation in the simulations. This would be discussed in detail in Chapter 6 and 7. We focus only on differences in rendering approximations in this chapter.

5.4.3 Impact of Photorealism

The $mIoU$ of DeepLab trained with simulated set, *Lambertian* (diffuse reflections only) is worse than the other choices. It is clear that diffuse material assumptions in simulations do not hold in most real world conditions. When we used the $Cook - Torrance$ shader that simulates both diffuse and glossy reflections the performance was worse. One possible explanation could be that the deep learned



(a) mean IoU vs rendering settings



(b) per-class IoU vs rendering

Figure 5.2: Performance (IoU on *CS_val*) variations due to photorealism and computational-approximations of rendering method

classifier system is not invariant to physical reflections. We, therefore, deactivated glossy reflections for this study. When we used more photorealistic (*RayTrace* and *mcpt*) sets with global illumination effects, the performance of trained model has been improved drastically (nearly gets double). Thus, it appears that photorealistic data might be necessary for a better generalization performance of trained DCNN models. But, how does the physical accuracy of these photorealistic effects impact the performance?

5.4.4 Impact of Physics fidelity

To know how physics fidelity in lighting computation impact the trained model, we compare the performance between these datasets (see Table 5.1). *mcpt_10* seems to be bad compared to *RayTrace*. However, 10 samples-per-pixel is too less to get a photorealistic image and it suffers from heavy sampling noise. Hence comparing with *mcpt_10* may not be fair. By comparing the performance of *RayTrace* and other *mcpt_X* ($X \geq 40$) sets, it appears that physics fidelity seems to be improving the performance a little bit at the cost of high rendering times. For example, on an average, the performance has been improved by 6% when trained with *mcpt* sets instead *RayTrace* set, but at the cost of more than 10 times the rendering time of ray-tracer (last column in the table). So, we believe that physics-accuracy of lighting computations of photorealistic effects is also important to some extent. However, it is a trade-off between large computational resources and little performance gains due to physics.

5.4.5 Impact of computational-approximation (spp)

We now examine the effect of MC sampling parameter (spp) in MCPT rendering method. Figure 5.2a is intended to show the variations in IoU due to photorealistic rendering and computational-approximation. The plot seems to be more or less flat for *mcpt* datasets, especially after *spp* = 40. The performance for *mcpt* sets varies between IoU $51.8 \pm 5.66\%$, mainly due to rendering noise in the simulations. From this experiment, we conclude that DCNN's seem to be less sensitive to computational-approximation of physics in simulated training dataset used in our application context and that 40 *spp* were enough for achieving the good performance in our experiments. Hence, accuracy seems to be important to some extent, but, large number of samples (*spp*) may not be necessary. This insight can help us to reduce the time required for rendering the data and be spent at some other means of performance improvement.

Table 5.2: Data augmentations and Generalization performance of DeepLab

Training	**Validation**	**Global**	**vehicle**	**pedestrian**	**building**	**vegetation**	**road**	**ground**	**sky**
CStrain	*CSval*	67.54	58.92	57.04	72.92	63.7	68.79	63.78	87.64
CStrain	*CamVidval*	54.29	47.33	42.58	54.84	68.67	45.52	51.25	69.89
mcpt40	*CSval*	50	48.49	53.39	63.34	50.78	46.66	34.27	53.09
mcpt40	*CamVidval*	39.37	52.97	28.14	34.26	35.18	42.73	19.47	62.85
mcpt40 + 10% *CStrain*	*CSval*	67.21	60.1	65.95	51.85	66.68	73.41	71.61	80.91

5.4.6 Things vs Stuff

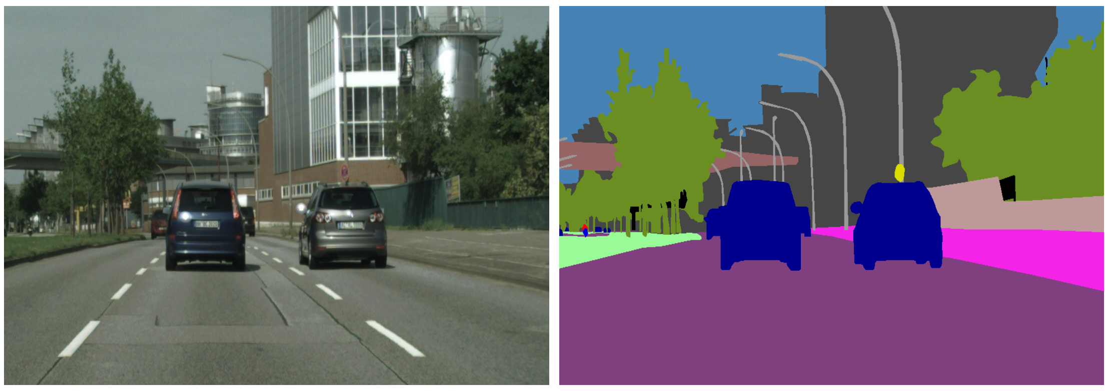
Figure 5.2b shows per-class IoU measures for all rendering settings, while the last row in Table 5.1 contains variances of per-class IoU measures of training settings with $mcpt\_X$, $X \geq 40$. One interesting observation from these numbers is that the computational-approximation does not seem to affect the system much in classifying the pixels of sky, ground, road (which are considered to be "Stuff[1]" in the semantic segmentation literature(please refer to Things-and-Stuff model[34]), while most of the differences are from the objects such as pedestrian, vehicles etc. (which are considered to be "Things" [34]). The reason we think is that the things (vehicles, pedestrians etc.) tend to have more diverse complex shapes and textures which may require more detailed CAD models and spp, compared to the stuff (building, ground, and sky etc.). This observation from object level analysis inspired us to analyze and locate the errors at pixel and region level.

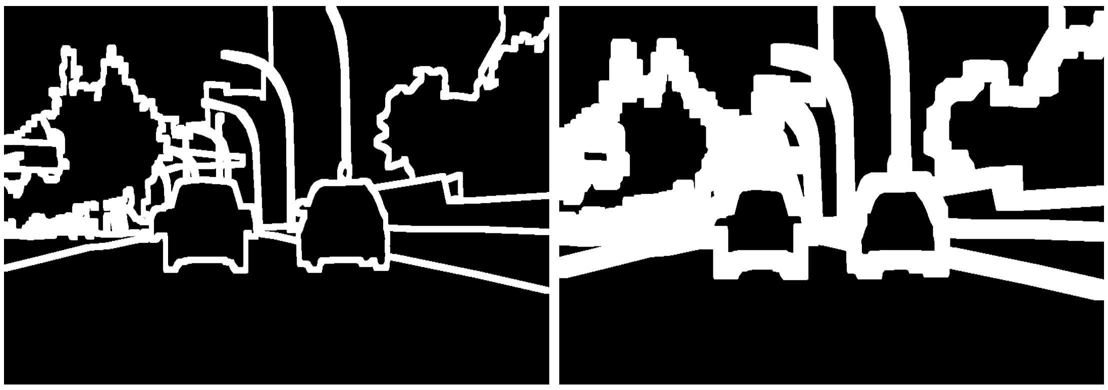
5.4.7 Location of Major errors

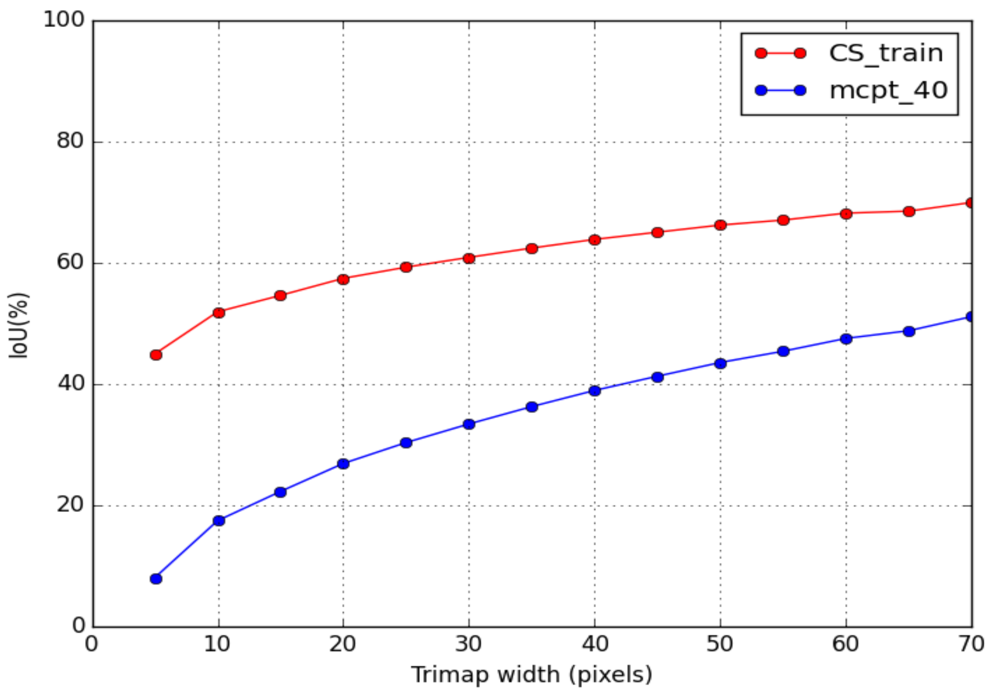
We inspect also to locate regions where virtual-world-based training results in classifications that are not reliable enough and deviate significantly. One key insight from the analysis of the input image statistics (c.f. Section 5.4.1) is that the simulated data might differ more at high-frequency contents such as inter-object boundaries. We would like to see how this deviation propagates through DeepLab training.

We approach this by analyzing the performance at object boundaries with the help of trimaps [41]. Trimaps are binary masks with the pixels that are located within a narrow band of object boundaries, as shown in Figure 5.3a. We create trimaps of varying pixel-widths for all images of CS_val, and compute $mIoU$ only at those pixels in the white band region. Figure 5.3c show how performance changes from boundaries to more global spatial contexts. One can observe that the performance deviates more near boundaries (lower values of trimap's pixel-widths) than that over entire image space. We postulate that this may be due to the same reason that the higher frequency contents more deviate from reality in simulated sets. The

[1]The objects that has no limited spatial extents are considered as *Stuff*, such as road, ground, sky etc., while *Things* represent the objects with limited sizes, such as pedestrians and vehicles.


(a) Input image and corresponding labels


(b) Trimaps of 10 and 30 pixel-width



(c) IoU vs Trimap width

Figure 5.3: Major erroneous locations: Major errors are located around object boundaries

difference between simulated and real sets can be explained by the fact that the object boundaries and shadows in virtual worlds are quite sharp while real world boundaries have effects of color bleeding and penumbra (due to sensor effects). So, modeling sensor and lens effects and behaviour in simulations or designing invarance these deviations in CV systems may be important to improve the generalization of the systerm and mitigate these statistical deviations. We address this issue in Chapter 8 in more detail.

5.4.8 Transfer learning and Data combinations

Our experiments demonstrated that physics based rendering might improve the generalization capabilities of DCNN models trained on simulated sets. However, one can see (in Table 5.1) that the performance of virtual-world-based-training is largely biased in all rendering settings compared to real-world training (see the difference between the blue curve and red line in Figure 5.2a). One possible explanation for this behaviour is that our virtual scene generative model might be biased to represent real-world geometric variations. We address this in the next chapters. However, this kind of domain shift issue exists in any two real-world datasets captured with different cameras or in different locations. For instance, please see Table 5.2. A DeepLab model trained on *CS_train* is achieving the performance levels of 69.54% on *CS_val* (which is from same benchmark set), while a performance degradation of 16% has been observed when it is tested on another validation set, *CamVid_val* [7] (captured in a different geographical location and with different camera model). Hence, we posit that virtual world datasets suffer from a dataset-shift problem just like any other real world dataset. The similar arguments and experiments have been demonstrated in the works [80]. To correct this bias, one can add a few real world samples to simulated training data as demonstrated in the works [78, 79, 31, 25, 65, 63]. The performance measures of different training-validation settings is shown in Table 5.2.

Like many works [78, 79, 31, 25, 65, 63] already reported, we also found that real world samples added to the simulated data can correct the performance bias of the trained model. In this experiment, we add 10% real world data samples of the training data from *CityScapes* and retrain the system. The performance of model trained on such augmented set has been improved by nearly 13% on *CS_val* set as shown in Table 5.2. These *mIoU* values are on par with the levels that are achieved with full training data. This can significantly reduce the number of real world samples needed at training/development phase.

5.5 Discussion and Conclusion

In this chapter, we addressed the very first fundamental question about the impact of rendering choices on virtual-reality based transfer learning in the context of semantic segmentation in traffic scenes. We empirically validated the virtual-world-based-training and provided several insights about the impact of photorealism and fidelity on DCNNs. Although it may be advantageous to have all photorealistic effects in the data, DCNNs are not too sensitive towards physics accuracy of the effects. However in the context of our experiments, physically based rendering seems to be improving the generalization performance by nearly 6% but at the cost of more than 3 times rendering time. Also, extreme levels of photorealism may not be necessary when Monte-Carlo based physics renderers are used for simulations. The virtual-world based training is relatively more biased around image boundaries due to fact that rendering methods may not be able to simulate complex phenomena around object boundaries. The major role in performance bias is from level-of-diversity of the virtual world model to capture real-world variations. In this aspect, virtual data suffers from a dataset shift problem just like any other real world dataset. However, this shift can be corrected by adding a few real world samples to training data. In our experiments, just 10% real world dataset was enough to reach the levels of full real world training. This significantly reduces the number of real world samples required at development phase.

6 Adversarially Tuned Scene Generation

In the previous chapter, we have evaluated the impact of rendering method and its parameters on the generalizability of virtual datasets to real-world. In the experiments reported, these sets were generated using uniform prior distributions on parameters of scene generative model (such as light source parameters, object scales, orientations and weather scattering parameters etc.). Such virtual data when simulated with physically based renderers was able to produce a DeepLab model which has shown a generalization performance of 50 $mIoU$ points on CityScapes (real-world) testing data. However, it is still nearly 17.54 $mIoU$ points less compared to the model trained on real CityScapes training set. We believe that this performance shift is due to the use of (uniform) prior distributions that may not accurately represent the target domain distributions (i.e. CityScapes). In principle, if one uses the distributions tuned to the target domain, then the simulated data will be more generalizable to tesing data from the target domain. Hence, in this chapter we propose an algorithm to estimate the prior distributions for the parameters of our scene generative model (c.f. Chapter 4) from a given unlabeled dataset from the target domain.

Training generative models is not easy in practice and still being an active research area. However, recent advance in the fields of unsupervised generative learning, i.e. *Generative Adversarial Training* [28] (popularly known as GANs, generative adversarial networks), proposes to use unlabeled samples from target domain to estimate generative model parameters (point estimates in statistics sense) by minimizing the discrepancy b/w generative and target distributions in the space of deep discriminatively-trained classifier. We adopt these concepts to tune parametric prior distributions in the context of CG based data generation scheme.

In the traditional GAN approach both generative and discriminative models are neural networks [28, 59]). In contrary, our work focuses on the iterative estimation of the posterior density of prior distributions, $P(\Theta)$, for our generative graphical model via:

1. generation of virtual samples given a starting prior,

2. estimation of conditional class probabilities of labeling a given virtual sample as real data using a discriminative classifier network (C),

3. mapping these conditional class probabilities to estimate class conditional probabilities for labeling of data as real given the parameters of the generative model (Θ),

4. finally, doing a Bayesian update to estimate the posterior density.

Initially, we assume uniform distributions as priors on these parameters of the generative scene model. As iterations proceed the uniform prior distributions get updated to distributions that are closer to the (unknown) distributions of target data. Please see Figure 6.1 for a schematic flow of our adversarial tuning procedure.

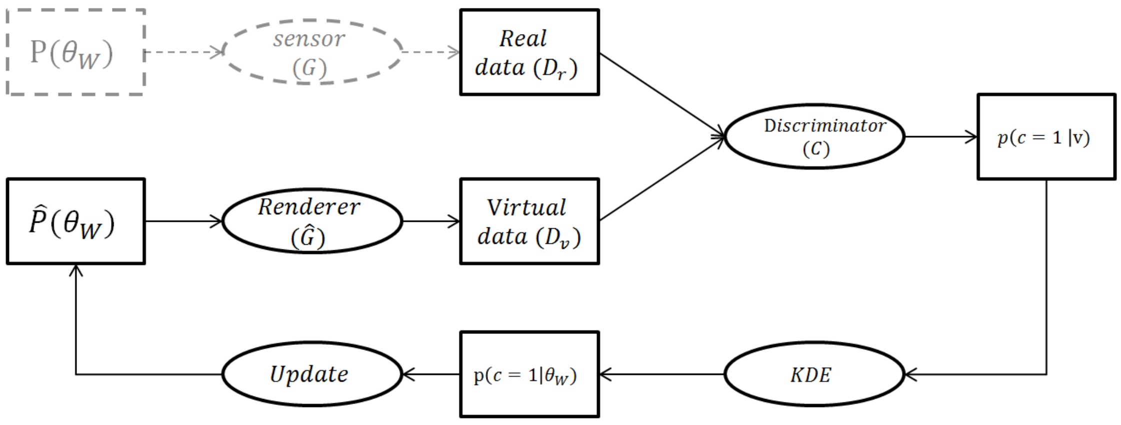


Figure 6.1: Flow chart of our adversarial tuning procedure: We sample 3D scene instances from the scene generative model and render the images. We then append a second discriminator network, C (a standard convolutional neural network) that tries to classify if an input image is real or synthetically generated. C is a discriminative classifier that has been trained on a combined dataset which has both simulated samples from the generative model and real samples from the target dataset. The classification probabilities across simulated instance, along with the parameter choices used in the simulation are aggregated together to get an estimate of the probability that a given choice of parameters produces near-target data. This likelihood model is used to update the parameters of $\hat{P}$ in a Bayesian update setting, such that it generates more similar images (as target data) in next iteration. In the end, uniform prior distributions are supposed to be updated to distributions that are closer to the (unknown) distributions of target data.

6.1 The Proposed Approach

More specifically, we use our parametric generative 3D scene model (G) as explained in Chapter 4. This model is parametrized by several variables including (a) *Light variables*:- intensity, spectrum, position of light source, weather scattering parameters; (b) *Geometry variables*:- object cooccurrences, spatial alignments; (c) *Camera parameters*: position, location of camera.

Our approach to tune a generative model to given target data was shown in Figure 6.1. We summarize the key steps below:

- The generative model ($\hat{P}$) has a set of parameters (θ_W) related to different scene attributes such as geometry and photometry etc.
- A renderer ($\hat{G}$) takes these parameters sampled from the distributions $\hat{P}(\theta_W)$ and outputs image data V.
- C is a standard convolutional network which can be trained with gradient descent on the data from target domain (T) and V to classify real vs generated. C outputs a scalar probability, which is trained to be high if the input was real and low if generated from G.
- The probabilities for all simulated samples $P(c = 1|v, \Theta) \forall v \in V$, are used to estimate a likelihood $P(v \approx real|\theta_W)$.
- This is then used to update our prior distributions, which will be used in the next iteration as $P(\theta_W)$.

We now describe the details of the components used in this process.

6.1.1 Initialization

Our scene generative model ($\hat{P}$) is explained in Chapter 4. As shown in Figure 4.1, our generative model is a physics-based parametric model whose inputs are a set of scene variables (θ_W) such as lighting, weather, geometry and camera parameters. We assume that all these parameters are independent to each other which provides the least expensive option for modeling and sampling. One can place the distributions on these parameters using expert's knowledge on the target domain or from other disciplines such as atmospheric optics, geographic and demographic studies. However, in the absence of priors, we can use uniform distributions in their permissible ranges. For instance, lights source's intensity is modeled as $uniform(low = 0, high = 6)$, *python notation*, where 0 intensity levels can correspond to night lighting while 6 is for lighting at noon. With these settings, our model was able to render physically plausible and visually realistic images. This scene model was used in our work in previous chapters (Chapter 4 and 5) and the performance of DeepLab model on the simulated data for semantic segmentation was quite generalizable to real-world. Yet, data-shift was observed due to deviations in scene generation statistics to target real-world domain. Hence, we now focus on the task of matching generation statistics to that of real world target data (for instance CityScapes [16]). Some of the samples rendered in this initial setting are shown in Fig 6.2b.

6.1.2 Sampling and Rendering

Although sampling from $\hat{P}(\theta_W)$ was easy initially, it eventually becomes hard as $\hat{P}$ gets updated iteratively in a Bayesian update fashion ($\hat{P} \leftarrow \hat{P}(\theta_W)p(.|\theta_W)$). We do not have conjugate relationships between classifier's probabilities and $\hat{P}(.)$. Hence, these intermediate $\hat{P}$ functions lose their *easy-to-sample-from* structure. Hence, we use rejection sampling scheme to sample from P due to its scalability. In general, an open issue in the use of rejection sampling schemes is to come up with an optimal scaling factor (M) which makes proposal distribution as an envelope for the complicated distribution that we should sample from. This issue does not arise in our case as our initial uniform distributions of $\hat{P}$ can behave as envelopes for all intermediate $\hat{P}$'s if they are not re-normalized. However, this ends up in the increase in the probability of rejecting many samples and generating samples becomes computationally heavy with the number of iterations. We solve this by normalizing intermediate probability tables with its maximum value. We used Monte Carlo path tracer in Blender to render the image samples (along with semantic labels groundtruth) from 3D scene states.

6.1.3 Adversarial Training

In an adversarial training setting, generation model is appended with a discriminator (C) which would be trained to classify real vs generated sample. In simple terms, the output of the discriminator (c) should be one for a real image and zero for a generated image. One can select any off-the-shelf classifier as C. However, the choice of C plays a critical role as it measures dissimilarity between $\hat{P}$ and P in the feature space that C is based on. Here we use AlexNet (5 layer convolutional neural net) as C to learn the feature space automatically as in conventional GANs. Standard stochastic gradient descent with backpropagation is used to train this net.

Training Domain Classifier (C): All images are resized to a common resolution of $223X223$ (default input size of Alex-net implementation in tensorflow). This is done to speed-up the training process and save memory. However, it has the disadvantage of missing the details of smaller objects such as pedestrians and vehicles. All real images (T) are labeled as 1, while simulated data is labeled as 0. Data augmentation techniques such as random cropping, left-right flipping, random brightness and contrast modifiers have been applied. per-image whitening has been used. 10000 epochs are used to train the classifier.

Tuning Scene Generator G: We now estimate the quantity $P(c = 1|\theta_W)$ from class probabilities (softmax outputs of C) of all virtual samples in V. This is estimated by weighted Gaussian kernel density estimation (KDE). Using classifier outputs ($p(c = 1|v)$) as weights:

$$P(c = 1|\theta_W) = \sum_{v \in V} p(c = 1|v) K_g(\theta_W^{(v)}, h) \tag{6.1}$$

where K_g a Gaussian kernel with bandwidth h. In our experiments, we use $h = 0.1$. We explored the use of automated bandwidth selection methods in the literature but in our experiments a default setting seemed to perform adequately. This KDE estimate represents the likeliness of $\hat{P}$ generating samples similar to T for given values of θ_W. In a Bayesian setting, this can be used to update our prior beliefs about $\hat{P}(\theta_W)$ iteratively as,

$$P^{(i+1)}(\Theta) \leftarrow P^{(i)}(c = 1|\theta_W) P^{(i)}(\theta_W) \tag{6.2}$$

After several iterations, if $\hat{P}$ and C have enough capacity, they will reach a point at which both cannot improve because $\hat{P}(\theta_W) \rightarrow P(\theta_W)$. The discriminator is unable to differentiate between the two distributions and becomes random classifier, i.e. $p(c) = 0.5$). However, we fix maximum number of $\hat{P}$ iterations to 6 in the following experimentation.

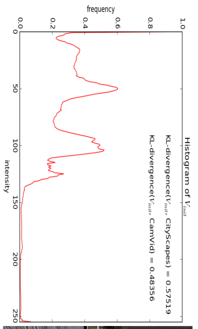
6.2 Validation

In this section, we provide an evaluation of our generative adversarial tuning approach by observing the data statistics simulated before and after tuning.

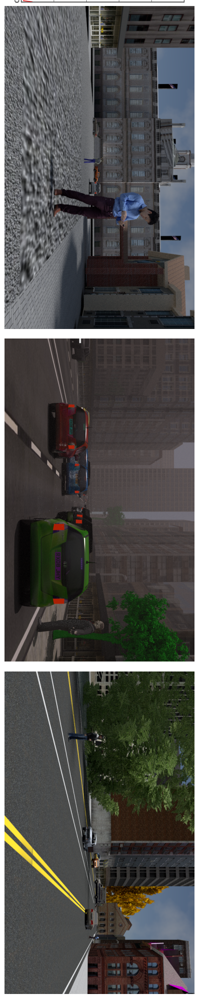
6.2.1 Real world target datasets

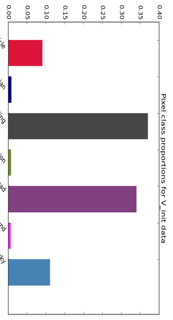
We used CityScapes [16] and CamVid [7] as target datasets which are tailored for urban scene semantic segmentation. CityScapes dataset was recorded on the streets of several European cities. It provides a diverse set of videos with a public access to 3475 images that has finer pixel-level annotations for semantic labels. However, in the

V_{init}



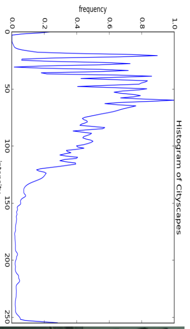
(a) Histogram of V_{init}


(b) A few samples of V_{init} sampled from the model before tuning



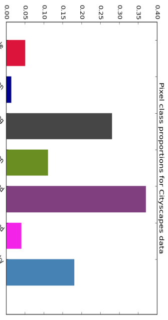
(c) Pixel-proportions/class

Cityscapes data



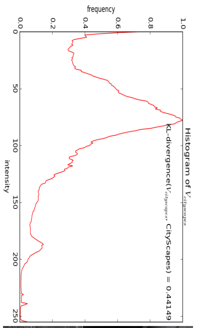
(d) Histogram of CityScapes

(e) A few samples from CityScapes data

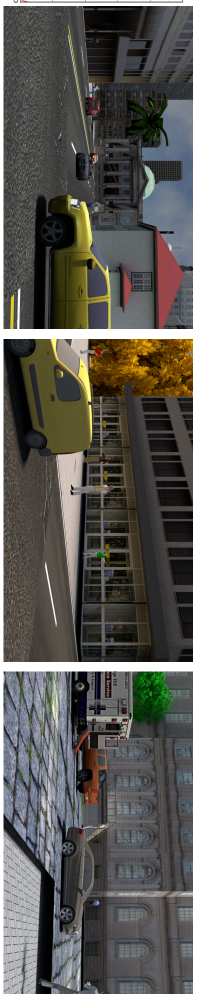


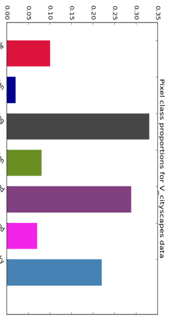
(f) Pixel-proportions/class

$V_{cityscapes}$



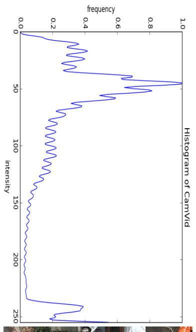
(g) Histogram of $V_{cityscapes}$


(h) A few samples of $V_{cityscapes}$ sampled from the model after tuning



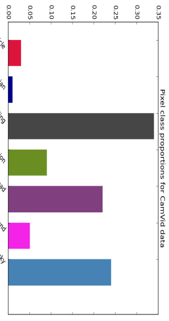
(i) Pixel-proportions/class

Camvid data



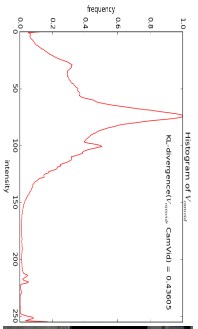
(j) Histogram of CamVid

(k) A few samples from CamVid data

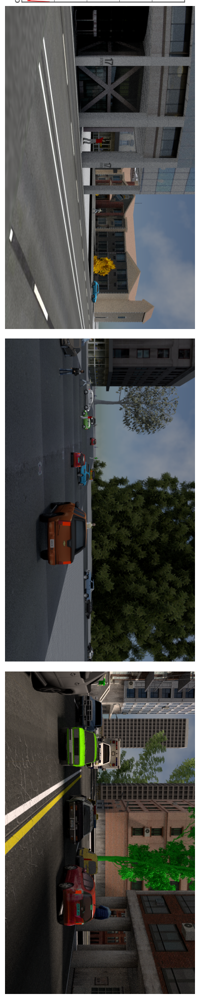


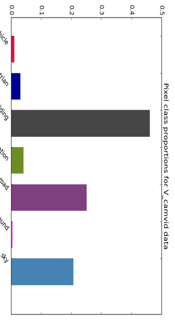
(l) Pixel-proportions/class

V_{camvid}



(m) Histogram of V_{camvid}


(n) A few samples of V_{camvid} sampled from the model after tuning



(o) Pixel-proportions/class

Figure 6.2: Qualitative comparison of training sets (both simulated and real) and their statistics before and after tuning

adversarial tuning process, we use 1000 randomly selected samples from CityScapes as T in each iteration to train D and we set $N_v = 1000$ to generate 1000 samples from $P(\Theta)$. CamVid is recorded in and around Cambridge region in the UK. It provides 701 images along with high-quality semantic annotations. While tuning the generative model to CamVid, we randomly sample 500 samples from CamVid in each iteration and set $N_v = 500$.

It is worth highlighting the differences between these datasets. Each of them has been acquired in a different city or cities. The camera models used are different. Due to the geographical and demographical differences in weather, lighting, object shapes, the statistics of these datasets may differ. For instance, we computed the intensity histograms over full CityScapes and CamVid datasets, (see Figure 6.2d and Figure 6.2j). For better visual comparison, we normalized the histograms with its maximum frequency. Topologically, these histograms are significantly different. Similarly label statistics also different (See the histograms of semantic class labels in Figure 6.2f and Figure 6.2l).

6.2.2 Virtual reality datasets

To realize the performance changes due to adversarial tuning, we prepared three sets that are simulated from the initial model and the models tuned (with the approach discussed in Section 6.1) to the datasets CityScapes and CamVid. We denote them with V_init, $V\_cityscapes$ and V_camvid respectively. Each set has 5000 images along with with pixel-wise semantic labels. We first compare the statistics of simulated training sets against the target datasets used for adversarial tuning. In next chapter, we also compare the generalizations of DeepLab models on the target dataset, when it is trained on these sets separately to quantify the performance shift due to adversarially trained scene generation.

6.2.3 Statistics of Training sets

Though its difficult to observe significant changes (due to tuning process) through visual inspection, in Figures (6.2b, 6.2h and 6.2n), we compared statistics of pixels and their labels to get an insight of what has been changed. We computed histograms of pixel intensities over the full datasets V_{init} (generated from the initial model), CityScapes (our target data) and $V_{cityscapes}$ (generated the model tuned to CityScapes). These plots are shown in the first column of Figure 6.2. Topologically, the structure of histogram has been moved closer to the histogram of CityScapes after tuning. Quantitatively, KL divergence between virtual data and CityScapes data has reduced from 0.57 (before tuning) to 0.44 (after tuning to CityScapes). Similar behaviour is observed when the model is trained to CamVid data. We also did similar kind of histogram analysis on the groundtruth labels. We observe these label statistics also are closer to real datasets after tuning as shown in the last column of Figure 6.2. This evidence points to the potential usefulness of simulated datasets as virtual proxies for these real world datasets.

7 Impact of Scene Generation Parameters

In this chapter, we address the other fundamental question: *How does parameter settings of scene generative model impact the generalization performance of DCNN models?*. We try to provide a quantitative answer by comparing the performances of DCNN models trained on three different virtual datasets separately. These sets are simulated using three different parameter settings (θ_W) of our scene generative model ($\hat{P}$), as explained in the previous chapter.

7.1 Generalization of DeepLab

As mentioned in the previous chapter, we simulate three sets: V_init, $V\_cityscapes$ and V_camvid with different parameter settings of the scene generative model. While V_init was generated by placing uniform distributions on scene parameters, $V\_cityscapes$ and V_camvid were generated after tuning those distributions to CityScapes and CamVid datasets respectively. We use these sets as training datasets for DeepLab and test their generalization performance on both CamVid and CityScapes datasets. The IoU performance metrics are tabulated in Table 7.1 with different training-testing combinations.

In our first set of experiments, we use CityScapes as the target domain which means we take validation set from CityScapes (CS_val) for testing. We compared the utility of simulated data generated from the initial model (V_{init}) and the model trained to CityScapes ($V_{cityscapes}$) in terms of the generalization of the trained models to CS_val. V_{init} produced good results in classifying the objects like building, vehicles, vegetation, roads and sky. However, pedestrians are poorly recognized due to low frequency of occurrences and use of low quality (low poly meshes and textured) CAD models. However, the use of $V_{cityscapes}$ (which is generated from the model trained to real CityScapes) has improved the overall performance (global IoU) by 2.28 points. This time, the per-class IoU measure on pedestrian class has also been improved by some extent. This may be credited to increased number of occurrences after tuning. This can be visualized in the bar plot of Figure 6.2 (last column). To measure

Table 7.1: Quantitative analysis of the performance of DeepLab models with different training-testing combinations.
Notation: CS and CV refers to real CityScapes and CamVid datasets respectively, and prefix 'V' represents simulated sets.

Training set	*Validation*	*global*	*vehicle*	*pedestrian*	*building*	*vegetation*	*road*	*ground*	*sky*
				Model Tuned to CityScapes data					
V_init	CS_val	49.86	48	53	63	51	47	34	53
V_cityscapes	CS_val	52.14 (+2.28)	56	47	65	57	53	31	56
CS_train	CS_val	67.71	59	57	73	64	69	64	88
V_cityscapes	CV_val	50.28 (+0.43)	51	50	55	48	49	49	50
CS_train	CV_val	54.42	47	43	55	69	46	51	70
				Model Tuned to CamVid Data					
V_init	CV_val	46.42	53	38	54	35	43	39	63
V_camvid	CV_val	49.85 (+3.42)	57	34	63	37	48	44	66
CV_train	CV_val	67.42	77	34	65	54	98	45	99
V_camvid	CS_val	39.85 (-6.57)	35	41	44	44	32	40	43
CV_train	CS_val	54.28	46	43	55	69	46	51	70
				Data augmentations					
V_init+10%CS	CS_val	67.42	60	66	52	67	74	72	81
V_cityscapes + 10%CS	CS_val	70.01 (+2.57)	68	60	59	68	77	69	89
V_init+10%CV	CV_val	68.85	51	61	71	67	65	77	90
V_camvid+10%CV	CV_val	70.57 (+1.71)	63	57	76	73	67	74	84

the statistical significance of these improvements, we repeated the training-testing experiment 5 times, measured the improvement each time. The computed mean and standard deviations are as 2.28 ± 0.34.

In our second set of experiments, we use CamVid as the target domain and take the validation set from CamVid (CV_val) for testing. We compared the utility of simulated data generated from the initial model (V_{init}) and the model trained to CamVid (V_{camvid}) in terms the generalization of the trained models to CV_val. V_{init} has already produced good results. However, the use of V_{camvid} has improved the overall performance (global IoU) by 3.42 points. Interestingly, the trained DeepLab model on $V_{cityscapes}$ has improved performance also on CamVid validation set, however vice-versa is not true as seen by a degradation in performance of 6.57%. We posit that the high number of pedestrian occurrences and their diversity in CityScapes set might be one of the reasons.

In the final set of experiments, we also consider a situation where 10% of real labeled training data is available from the corresponding target domain. Here, we compared the results of unsupervised adversarial tuning with those of supervised domain adaptation, i.e., augmenting the simulated with 10% labeled samples from the target domain. Clearly and obviously, supervised domain adaptation provides improved gains over our adversarial tuning approach. However, we note that our modest improvements using unsupervised learning described previously were achieved without labeled samples from target domain, thus, the cost of the improvement is cheap. Instead of using the data simulated with the initial model (V_{init}), if we use data from tuned models ($V_{cityscapes}$ and V_{camvid} for DeepLab training, we improve the performance (on corresponding validation sets) by 2.57 and 1.71 IoU points respectively. This gives an insight that the amount of real world labeled data required to correct domain-shift in order to achieve a similar level of performance as V_{init}+10%CS is reduced. A rough analysis (using a linear fit for performance gains) of the performance numbers in Table 7.1 provides the observation that the amount of labelled real world data needed to reach the same level performance with $V_{cityscapes}$ is 9% of training data compared to the 10% labeling of training data needed for V_{init}.

7.2 Conclusions

In Chapter 6 and 7, we have evaluated an adversarial approach to tune generative scene priors for the process of CG based data generation to train CV systems. To achieve this goal, we designed a parametric scene generative model, followed by AlexNet whose output probabilities are used to update the distributions of scene parameters. Our experiments in the context of urban scene semantic segmentation of with DeepLab provided evidence of improved generalization of models trained on simulated data generated from adversarially tuned scene models. These improvements were found to be 2.28% and 3.42% (average IoU) points on two real world benchmarking datasets, CityScapes and CamVid respectively.

Our current work does not vary the intrinsic attributes of objects (shapes and textures). Instead, we used a fixed set of CAD shapes and textures as a proxy to model intra-class variations. We think that the performance boosts are not significant as expected as this fixed CAD model set is restrictive. A possible extension is to use component-based shape synthesis models (similar to [40]) to learn distributions on object shapes. Further experimentation is needed to characterize the behavior of adversarial tuning by studying the variability of the performance on simulated training and target domains and plot the performance gains as a function of the KL-divergence between the prior distributions used for training and target domains.

8 Learning to Transfer Geometric Context

In this chapter, we propose an unsupervised methodology that uses "unlabeled" data from a target real-world domain to mitigate domain-shift problems of virtual data. Chapter 5 and Chapter 7 evaluated the role of rendering fidelity and tuning the scene model parameters respectively to deal with the domain-shift issues. Experimental results in those chapters showed that one could achieve better generalization capabilities for vision models trained on simulated data by adapting the generative models in scene space. Although it has resulted in improved generalization performance, we had to use a few "labeled" real samples (at least 9% of CityScapes in the context of experiments conducted in Chapter 7) to achieve the performance levels on par with that of full real-world training. Fortunately, we had access to semantic label information of real data in the case of semantic segmentation on CityScapes benchmark. However, this 9% data (i.e., more than 300 images) still requires a lot of human efforts to label at pixel-level, for instance, optical flow and intrinsic images, etc. By motivating the fact that unlabeled data is abundant in real-world, this chapter explores the use of unlabeled real data in a Multi-Task Learning (MTL) [67] framework to reduce the domain shift of the feature representations that are learned while training the vision models on labeled simulated data.

As we have seen in Chapter 5, the appearance of simulated data largely depends on modeling errors and computational approximations underlying the chosen rendering algorithm. Hence, appearance statistics of simulated training sets might be severely biased unless advanced physically realistic rendering algorithms used are closer to the sensory processes in the real-world camera used to capture real resting data. We believe that the object shape and scene geometry models are more realistic in virtual worlds (especially in modern graphics tools that use high polygon 3D meshes and well-sampled animation models). Thus, the geometric context-based features are relatively less biased compared to appearance models. Hence, we hypothesize that transfer learning from virtual-to-reality would be more effective if we can regularize the process of training on simulated data to learn geometric contextual features rather than appearance-specific features. Towards this end, we design a training methodology to learn the features that are not only to solve the given task (i.e., semantic segmentation on labeled simulated data) but also solve an auxiliary task (on real data) that requires an understanding of the geometric context of images. We believe that training two CNN's with shared components to solve the given task and auxiliary geometric problem will produce features that are more robust towards domain discrepancies b/w virtual and real-worlds.

However, it is more difficult to get ground truth labels for most of the geometry context prediction tasks. Moreover, it is not very meaningful to select a task that requires labels that are harder to collect than the semantic labels. Inspired by recent concepts of self-supervised learning [60], we design a task and training methodology that does not require any manual labels on training data and obtain supervisory training signal from the image itself. Also, solving this task should require a semantic contextual understanding of the entire image instance. These requirements dictated us to choose "**inpainting**" task as the auxiliary task as it is easy to generate training data from real images without any manual intervention. To succeed at inpainting task, the network has to understand geometric context of entire image as well as produce plausible hypotheses for the missing regions. This requires a more in-depth semantic understanding of the scene, and ability to synthesize high-level features over large spatial extents.

We start by randomly masking out some regions of given real image. We then train a CNN (namely InpaintNet) to regress the missing pixel values. In this work, we use encoder-decoder like architectures for both inpainting and semantic segmentation tasks. We use a fully convolutional network for semantic segmentation that is adapted from the implementation of SegNet proposed in [5]. We train SegNet on labeled simulated data and InpaintNet on unlabeled real data, both simultaneously in an MTL framework. MTL frameworks encourage the networks to learn the features representations that are generalizable to both the tasks, thus, both the domains. Hence, SegNet features that are learned from simulated data would be well generalizable for real data also.

In general, one can train InpaintNet using reconstruction losses like L2/L1 norms [12]. However, they tend to produce blurry results due to inherent multi-modal nature of the inpainting task, especially in the case of large missing regions. In other terms, there are multiple plausible hypotheses to fill the missing regions while maintaining coherence with the surrounding spatial context. Hence, predicting the mean of this multi-modal distribution of hypotheses results in minimum for reconstruction losses like L2 loss. To overcome this problem, we add an adversarial discriminator that helps InpaintNet to produce high quality and visually sharper regions.

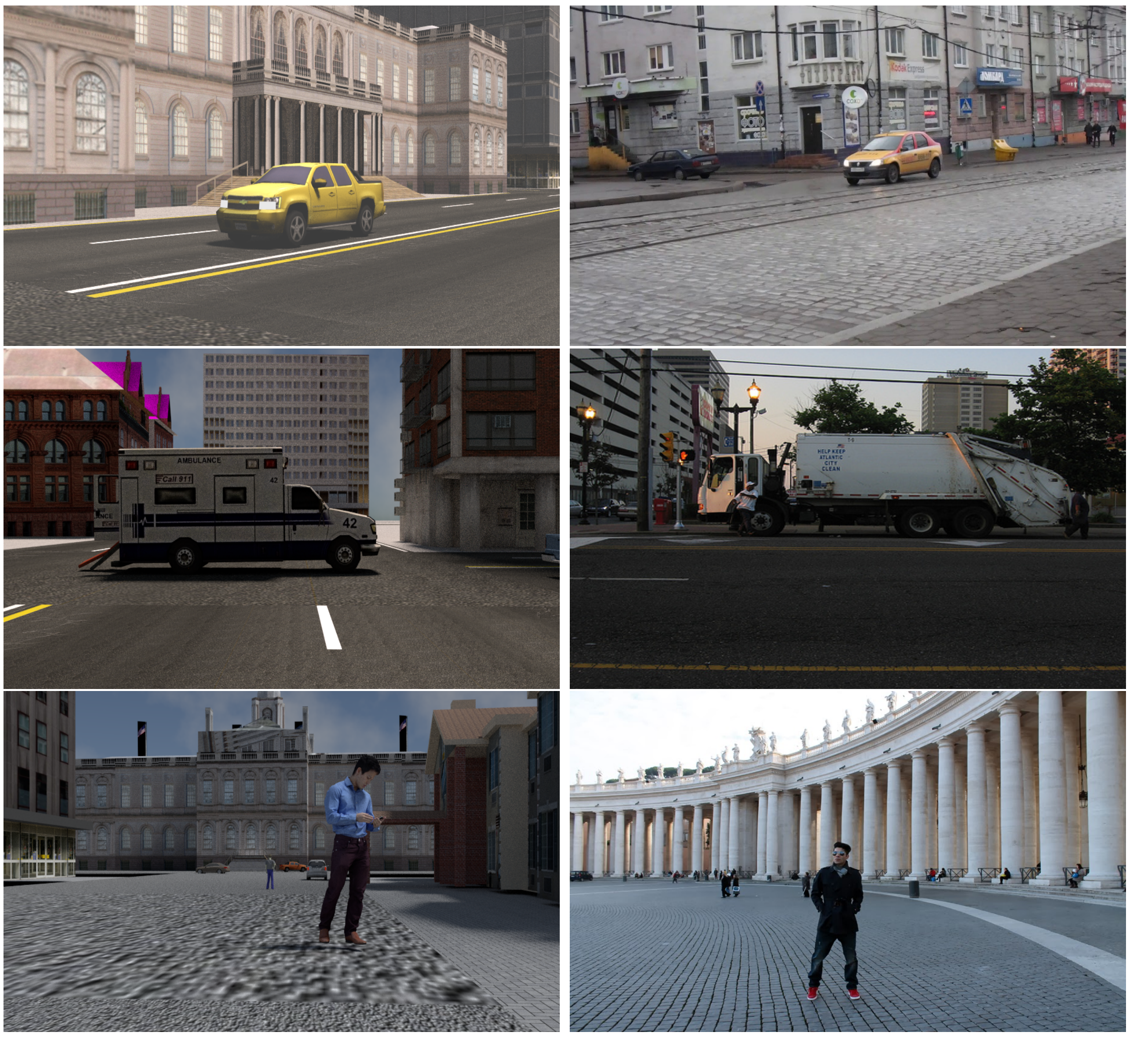


Figure 8.1: Geometric context might be less biased across virtual and real traffic scenes compared to appearance context. For instance, geometry scene layouts and object relations seem very similar across simulated (left column) and real (right column) scenes, while the differences in appearance of the images are quite varying.

8.1 Background

In this section, we provide a brief review of the concepts that are related to the work in this chapter.

8.1.1 Force to Learn Geometric Context

Geometric (both spatial and temporal) context is the most realistic content in the graphics-generated images/videos than appearance context [56]. Thanks to high-poly meshes, textures, and animations in modern graphics tools. For instance, please refer to Figure 8.1, distributions of scene geometry and object shapes are very similar between virtual and real worlds, especially in the context of traffic scenes, while their appearance might vary significantly (depending on rendering engine used). Also, as shown in the bottom row of Figure 8.1, object appearance, especially human avatars, in CG-based simulations are more biased in the appearance compared to their shape statistics. Hence, we believe that the vision models would be more effective in real-world conditions if training process on simulated data is regularized to learn and transfer the features that are based on geometric statistics rather than the appearance. Recent progress in multi-task learning paradigms for deep neural networks has proven that training two or more CNNs for multiple tasks but with shared weights improves the generalization capabilities of the trained models across tasks and domains. Training process in MTL can help the models to focus on the feature representations that more generalizable as other tasks provide additional evidence for the relevance or irrelevance of those features [67]. Motivated by these concepts, we propose an MTL based framework to train our semantic segmentation on labeled simulated data along with an auxiliary task on unlabeled real-data in a self-supervised manner.

However, one can come up with several tasks that require an understanding of the geometric context of the entire image. Examples include pose estimation [10], 3D object detection [45] etc. However, most of these tasks require training data to have corresponding output labels. This might require additional manual efforts since we aim to solve this task on real data. Alternatively, one can choose reconstruction tasks such as inpainting, etc. for which supervised training sets can be created by artificially applying geometric distortion on real data without any human efforts.

8.1.2 Inpainting

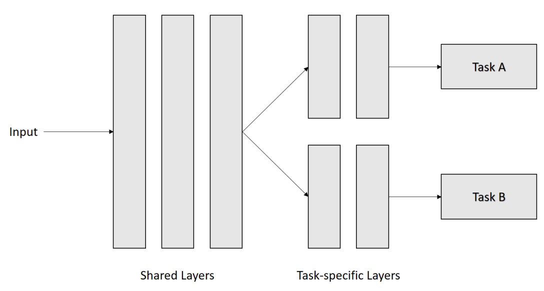
Image inpainting refers to the task of inferring arbitrary missing regions in images. In the case of large missing regions, the problem is termed as semantic inpainting as the algorithms have to understand the semantic context of the entire image to infer plausible hypotheses for missing regions. Since the prediction of semantic context is required, this task is significantly more difficult than classical hole filling algorithms which are more concerned with correcting spurious data corruption. However, inpainting becomes increasingly more difficult if large regions are missing or if scenes are complex. Hence, we are interested in the more context-driven task of semantic inpainting instead of filling small holes in the image.

Auto-encoder like architectures have been proposed for the problem and trained using reconstruction losses such as L2 or L1 norms [84]. However, these losses are found to be producing blurry results. This might be because this task is inherently multi-modal as there are multiple ways to fill the missing region while also maintaining coherence with the given context. Hence, it is much safer for the L2 loss to predict the mean of the distribution because it minimizes the error (mean of the pixel-wise errors), but results in a blurry averaged regions. In our work, we try to alleviate this problem by adding an adversarial loss as would be explained in Section 8.2.

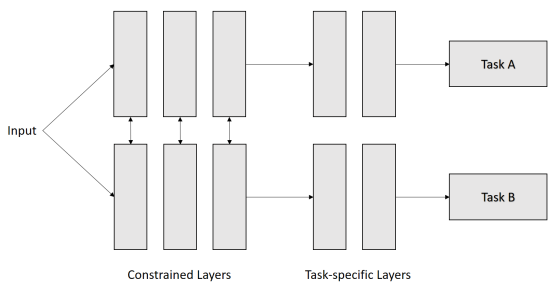
A recent approach for semantic inpainting, and closest to our work is [55]. This work uses the concepts of adversarial learning to encourage the network to predict in sharper regions. Given a mask indicating missing regions, an auto-econder like neural network is trained to encode the spatial context information and predict the unavailable content. Decoder's predicted region is then fed to discriminator along with some arbitrary real images sampled from the training set. The job of the discriminator is to classify real verses predicted regions. The entire network is trained in using a GAN loss. However, the decoder does not have enough prior information about the content in the missing region. Hence, it still results in blurry or unrealistic images especially when missing regions have arbitrary shapes. In our work, we solve this problem to some extent by sharing the encoder's weights of InpaintNet with that of segmentation network. The encoder's information learned from simulated data will act as a prior for InpaintNet. We also observe that sharing weights of the discriminator with the encoder of InpaintNet would stabilize the training process and produce very realistic hypotheses for missing regions.

8.1.3 Self-supervised Learning

Very recently, there has been significant interest in self-supervised learning processes [60] that aim to learn meaningful representations in supervised manner by automatically extracting supervisory signals implicitly present in the given input signal. Most of the works apply some arbitrary deformations to the input signal and train a CNN to reconstruct the ideal image or information about it. The trained CNN model



(a) hard parameter sharing



(b) soft parameter sharing

Figure 8.2: MTL paradigms

is then fine tuned to the given task using a few samples with task-specific labels. Some of the self-supervised tasks explored are jigsaw puzzles (prediction of relative spatial locations of patches) [19], video frame sorting (prediction of correct ordering of frames in a video) [44], missing color frame reconstruction [19], etc. However, CNN's can solve these tasks by learning to detect some low-level image properties that may not be relevant to semantic tasks [19]. On the other hand, inpainting with large missing regions could be a suitable task for our requirements on the auxiliary task, since it requires an understanding of the geometric context of the given image.

8.1.4 Multi-Task Learning

As mentioned above, we train two CNN's, SegNet and InpaintNet, simultaneously within a MTL framework. In the context of Deep Learning, MTL is typically done with either hard or soft parameter sharing of hidden layers.

Most common approach for MTL in deep networks is hard parameter sharing and initially explained in [11]. As shown in Figure 8.2a, hard parameter sharing approach is applied by sharing some lower level layers between all tasks while keeping separate task-specific output layers. It reduces the risk of overfitting. In fact, [6] showed that the risk of overfitting the shared parameters is an order N where N is the number of tasks smaller than overfitting the task-specific parameters, i.e. the output layers. This makes sense intuitively: The more tasks we are learning simultaneously, the more our model has to find a representation that captures requirements of all of the tasks and the less is our chance of overfitting on our original task.

In soft parameter sharing, on the other hand, each task has its own model with its own parameters. The distance between the parameters of the model is then regularized to encourage the parameters to be similar, as shown in Figure 8.2b. The work of [21] for instance use L2 distance for regularization, while [86] use the trace norm. Since our both the tasks are of different flavors, we use soft-parameter sharing with L2 norm as in [21].

If a task is very noisy or data is limited and high-dimensional, it can be difficult for a model to differentiate between relevant and irrelevant features. MTL can help the model focus its attention on those features that actually matter as other tasks will provide additional evidence for the relevance or irrelevance of those features. MTL biases the model to prefer representations that other tasks also prefer. This will also help the model to generalize to new tasks in the future as a hypothesis space that performs well for a sufficiently large number of training tasks will also perform well for learning novel tasks as long as they are from the same environment [6]. Finally, MTL acts as a regularizer by introducing an inductive bias. As such, it reduces the risk of overfitting as well as the Rademacher complexity of the model, i.e., its ability to fit random noise.

8.2 Proposed MTL Architecture

We now introduce our MTL architecture that can be trained simultaneously for both semantic segmentation (in a supervised manner) and inpainting (in a self-supervised manner). Since our MTL architecture has several components including SegNet, InpaintNet, and an adversarial discriminator, we first describe the SegNet part of the architecture that is supposed to solve semantic segmentation on simulated data. We then explain the design of InpaintNet that aims to solve inpainting task on unlabeled real data. We also describe the adversarial discriminator that encourages the InpaintNet to predict sharper and visually plausible hypotheses for missing regions in real image data. A schematic flow chart of MTL architecture is as shown in Figure 8.3.

8.2.1 SegNet with Instance Normalization Layers

We choose to develop our semantic segmentation architecture by adapting the SegNet implementation proposed in [5]. The reason this choice is that its encoder-decoder like structure (see Figure 8.3) provides an easy way to share weights with InpaintNet which also has similar architecture. Its encoder-decoder design is based on the convolutional layers of VGG-16 from the Visual Geometry Group [70]. In the original implementation of SegNet [5], encoder is a succession of convolutional layers followed by batch normalization (BN) [35] and ReLU (rectified linear units) layers. The motivation behind using BN layers is to reduce covariate shift across mini-batches. However, in this work, we work with minibatch samples come from two domains

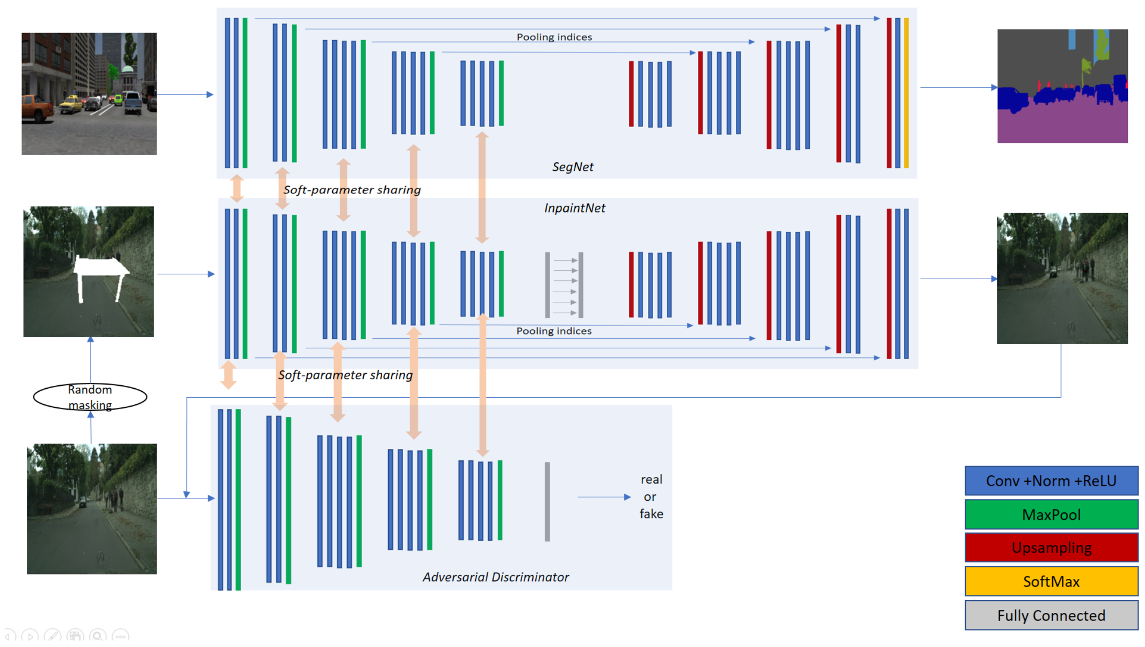


Figure 8.3: Our Multi Task Learning (MTL) architecture has three major components: SegNet that is supposed to solve semantic segmentation on simulated data, InpaintNet that aims to solve inpainting task on unlabeled real data, and an adversarial discriminator that encourages the InpaintNet to predict sharper and visually plausible hypotheses for missing regions in real image data.

(virtual and real) that may have a different kind of statistics. Hence, we replace BN layers with instance normalization (IN) layers [75, 20] which use per-feature statistics (mean and variance) to normalize the feature responses from convolutional layers. One more advantage with IN layers is that, unlike BN layers, they don't have to be removed at testing phases. Blocks of convolutions are followed by a pooling layer of stride 2. The decoder has the same number of convolutions and the same number of blocks. In place of pooling, the decoder performs upsampling using unpooling layers. This layer operates by relocating at the maximum index computed by the associated pooling layer. For example, the first pooling layer computes the mask of the maximum activations (the ”argmax”) and passes it to the last unpooling layer, which will upsample the feature map to a full resolution by placing the activations on the mask indices and zeroes everywhere else. The sparse feature maps are then densified by the consecutive convolutional layers. The encoding weights are initialized using the corresponding layers from VGG-16, and the decoding weights are initialized randomly using the strategy from [33].

8.2.1.1 Train SegNet on Simulated Data

We train our SegNet on simulated data since we have access to pixel-level labels. We use the cross-entropy loss as follows. Let K denote the number of semantic classes, which is 7 (similar to the ones in previous chapters) in our experiments. And, B be the number of images in a mini-batch and N the number of pixels in the input image. For a pixel x^i, let y^i be its ground truth label and $SegNet(x^i) = (z_1^i, z_2^i, ..., z_K^i)$ is prediction vector. We minimize the loss:

$$L_{seg} = \frac{1}{NB} \sum_{b=1}^{B} \sum_{i=1}^{N} \sum_{k=1}^{K} y_k^i(x_b) \log(z_k^i(x_b)) \tag{8.1}$$

8.2.2 InpaintNet

For Inpaint task, we mask out some region(s) from a real image using random binary masks. We use binary masks from PASCAL-VOC segmentation dataset and randomly apply for images from CityScapes as shown in Figure 8.4. Given an image with a missing region (e.g., left-column in Figure 8.4), we train a convolutional neural network (InpaintNet) to regress to the missing pixel values. It shares a similar encoder-decoder architecture as SegNet. Hence, it is trivial to share weights via soft-parameter sharing in MTL. Since inpainting is fundamentally a regression task, we replace last softmax layer with another convolutional layer with Tanh nonlinearity. Moreover, if the encoder architecture is limited only to convolutional layers, there is no way for information to directly propagate from one corner of the feature map to another. This is so because convolutional layers connect all the feature maps together, but never directly connect all locations within a specific feature map. We follow the work of [55] to alleviate this problem by using channel-wise fully connected layers followed by the encoder layers as shown in Figure 8.3.

Binary Masks: The input to InpaintNet is a real image (real target domain, for instance, CityScapes) with some region masked out; i.e., set to zero, assuming zero-centered inputs. The removed regions could be of any shape. The simplest such shape is the central square patch in the image. While this works quite well for inpainting, the network might learn low-level image features that latch onto the boundary of the rectangular or square mask [55]. Those low-level image features tend not to generalize well to images without masks. Hence the features learned are not very general.

To prevent the network from latching on the constant boundary patterns of the masked region, we randomize the masking process. We experimented with removing arbitrary shapes from images, obtained from random masks in the PASCAL VOC

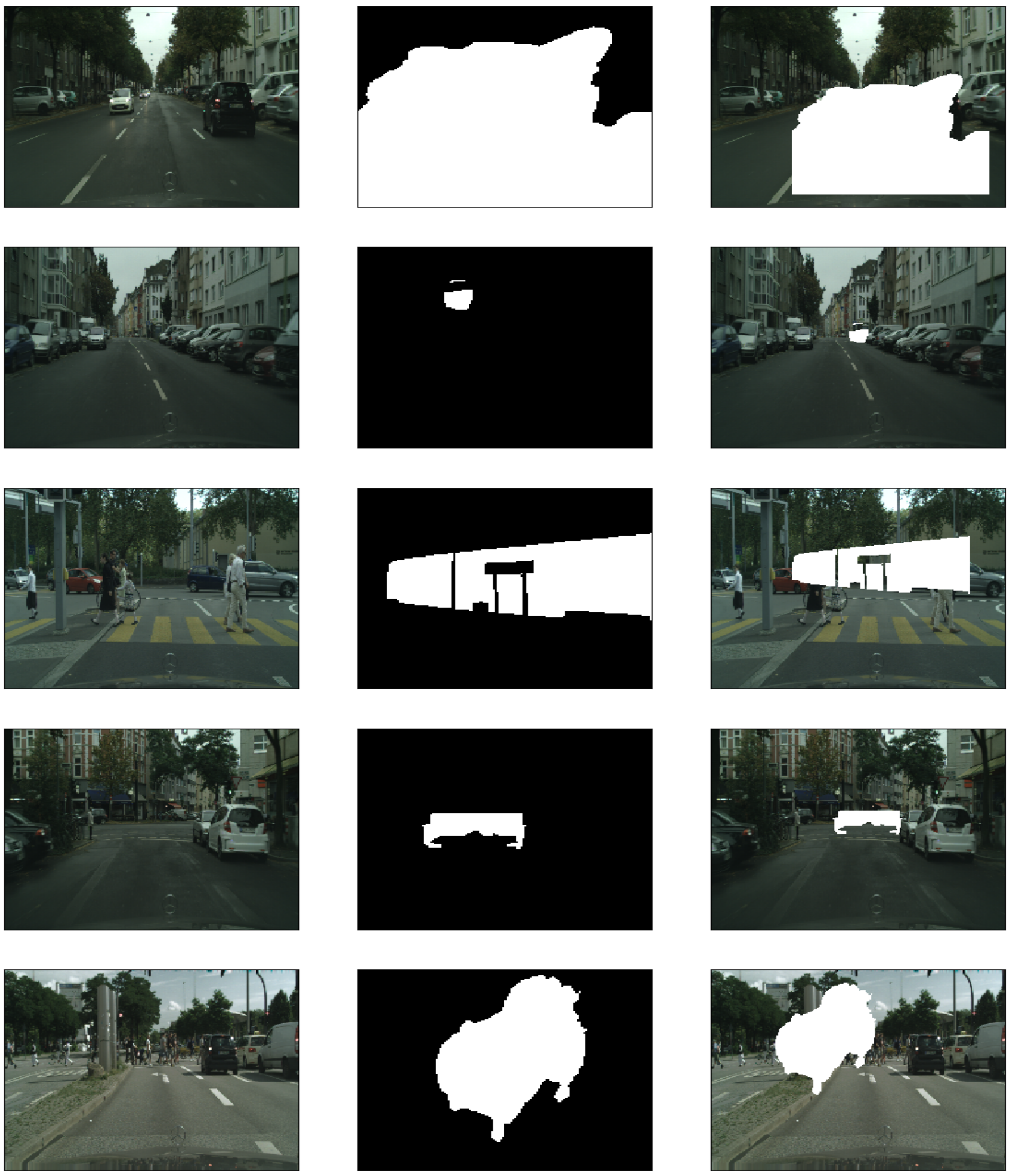


Figure 8.4: Input and ground truth generation for Inpainting task. Left: real images from CityScapes dataset, Middle: Binary masks randomly selected from PASCAL-VOC segmentation set, Right: Masked images which are inputs to InpaintNet and images in left column are used as ground truth for training.

2012 dataset [22]. We deform those shapes and paste in arbitrary places in the images (from CityScapes). Note that we completely randomize the region masking process, and do not expect or want any correlation between the source segmentation mask and the image. We merely use those regions to prevent the network from learning low-level features corresponding to the removed mask. See examples of training samples in Figure 8.4.

Loss: We use a normalized masked L2 distance as our reconstruction loss function,

$$L_{rec} = ||(1 - M) \odot (Y - X)||^2 \tag{8.2}$$

where M is mask used on original image, X, to generate input to InpaintNet ($M \odot X$) and Y is the predicted output. While this simple loss encourages the decoder to produce a rough outline of the predicted object, it often fails to capture any high-frequency detail. This stems from the fact that the L2 loss often prefers a blurry solution, over highly realistic textures. We believe this happens because it is much safer for the L2 loss to predict the mean of the distribution because this minimizes the mean pixel-wise error but results in a blurry averaged image. We alleviate this problem by adding an adversarial loss as follows.

8.2.3 Adversarial Discriminator

Inspired by recent advances in Generative Adversarial Networks (GAN) [28], we use an adversarial discriminator (D) to help our InpaintNet to synthesize realistic hypotheses for missing regions. The objective of D is discriminate b/w generated image and real image. The trainning process is a 2-player game when D takes both the prediction of InpaintNet and groundtruth samples and tries to distinguish them, while InpaintNet tries to fool D by predicting missing regions that appear as ”real” as possible. D is implemented with another CNN that is similar to the encoder part of InpaintNet or SegNwet. We also observed results improved when the encoder part of InpaintNet and D share the weights softly. The adversarial loss of D, L_{adv}, is given as

$$L_{adv} = max_D \mathbb{E}_x[log(D(X)) + log(1 - D(Y))] \tag{8.3}$$

Where, in practice, both InpaintNet and D are optimized jointly using alternating SGD. Note that this objective encourages the entire output of the InpaintNet to look realistic, not just the missing regions as in Equation 8.2.

8.2.4 Multi-task loss

Since inpainting and segmentation tasks are of different flavors, we use soft parameter sharing as MTL paradigm. The objective function for soft-parameter sharing based MTL can be written as

$$L_{soft} = \sum_{c \in C_{enc}} \Omega(W_{seg}^{(c)}, W_{inp}^{(c)}, W_{adv}^{(c)}) \tag{8.4}$$

where C_{enc} denotes all convolutional layers in encoder parts and $W_s^{(c)}$ represents vectorized form of all weights in a convolutional layer c of network s. $\Omega(W)$ is a regularizer that couples the learning problems, typically by encouraging $W = [W_{seg}^{(c)}, W_{inp}^{(c)}, W_{adv}^{(c)}]$ to be a low-rank matrix. Popular choice is to use L_2 norm [58]. Hence, we use

$$L_{soft} = \sum_{c \in C_{enc}} |W_{seg}^{(c)} - W_{inp}^{(c)}|^2 + |W_{inp}^{(c)} - W_{adv}^{(c)}|^2 \tag{8.5}$$

We define the overall MTL loss function as

$$L_{mtl} = \lambda_{seg} L_{seg} + \lambda_{inp} L_{inp} + \lambda_{adv} L_{adv} + \lambda_{soft} L_{soft} \tag{8.6}$$

8.2.5 Training

The pipeline was implemented in PyTorch [54]. We used the recently proposed stochastic gradient descent solver, ADAM [23] for optimization. The missing region in the masked input image is filled with constant mean value. For SegNet, we used the default solver hyper-parameters suggested in [5]. We use $\lambda_{seg} = \lambda_{inp} = \lambda_{soft} = 0.333$ and $\lambda_{adv} = 0.001$. However, a few things were crucial for training the model. We use a higher learning rate for InpaintNet (10 times) to that of adversarial discriminator. To further emphasize the consistency of prediction with the context, we predict a slightly larger patch that overlaps with the context (by 7px). During training, we use higher weight (10X) for the reconstruction loss in this overlapping region

8.3 Experiments

As baselines, we start by reporting the generalization performance (in terms of mIOU points) of SegNet models are trained separately on real and virtual data; and combinations of them. We then train SegNet in the above proposed MTL framework and compare the performance of those models with the baselines. Similar to previous chapters, we use validation split (CS_val) from CityScapes as our testing

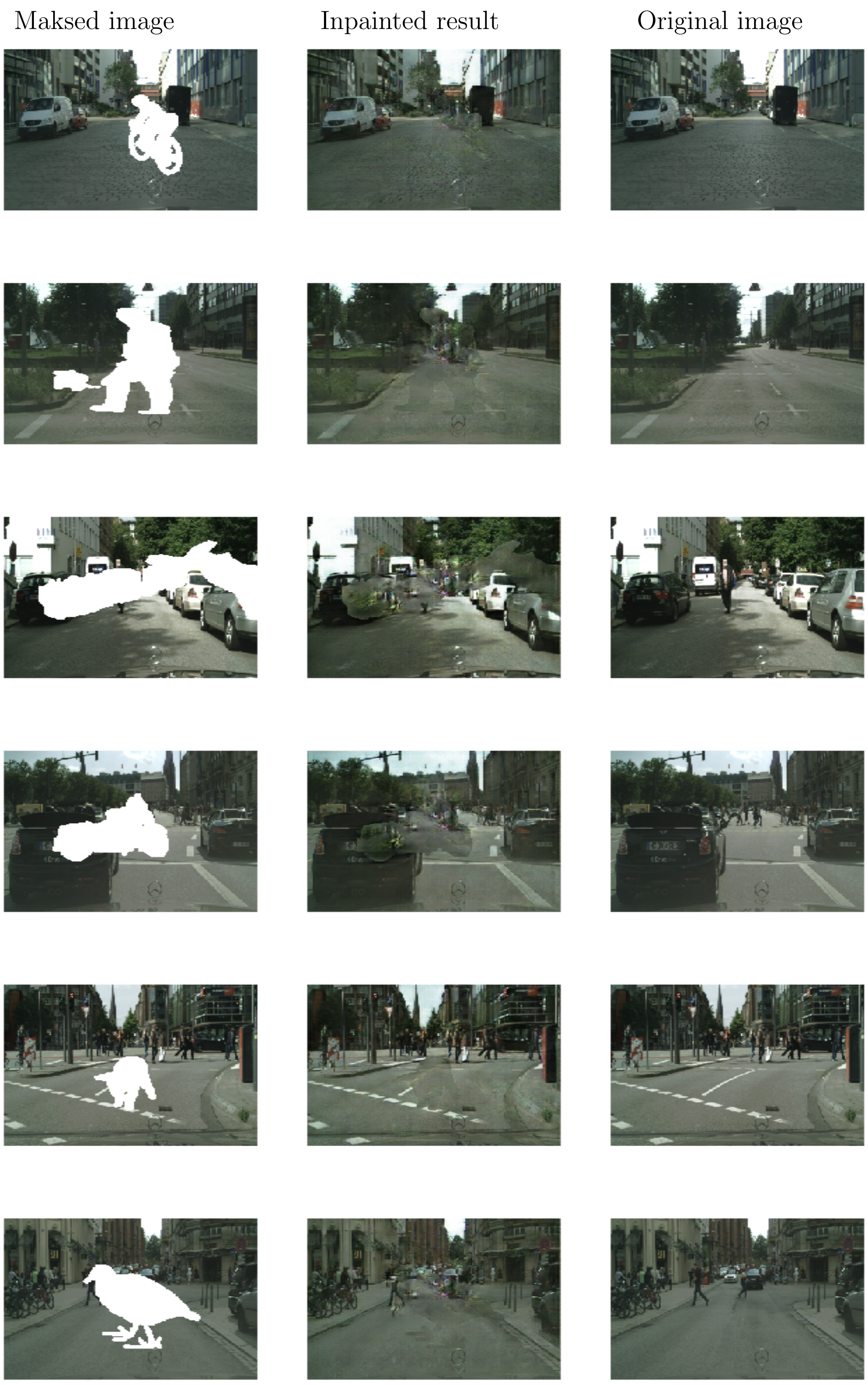
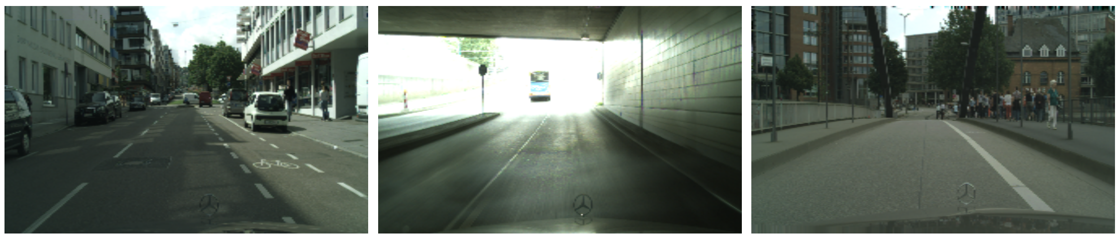
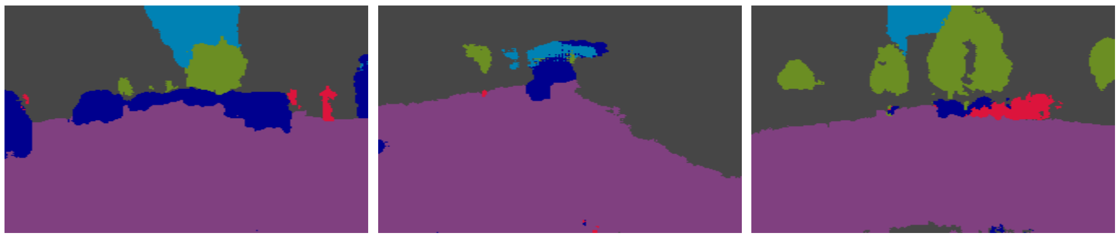
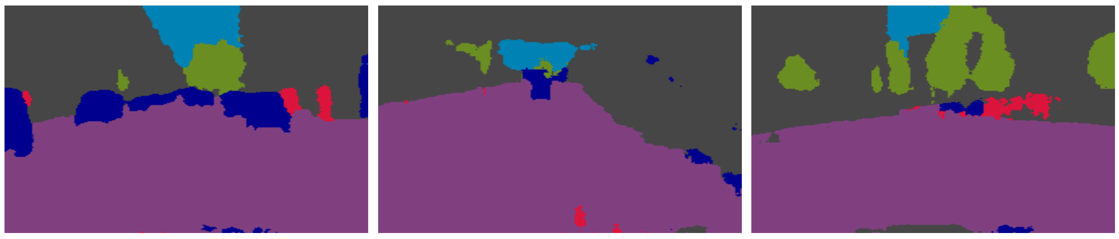
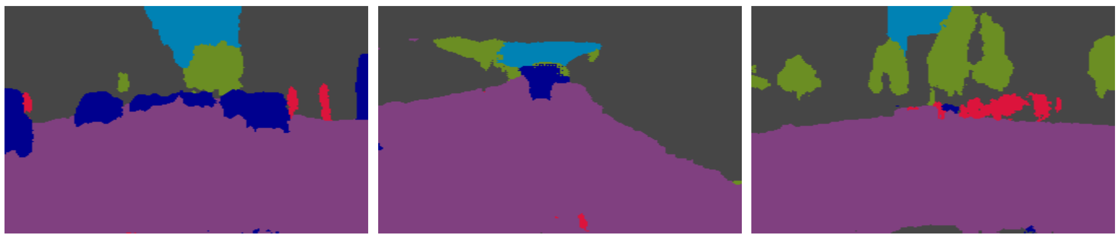


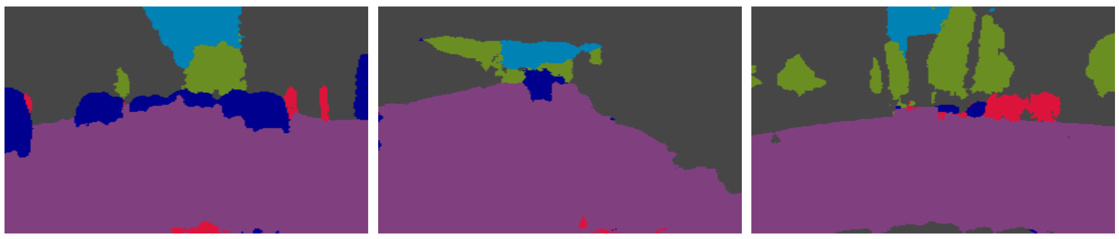
Figure 8.5: Inpainting results


(a) A few input images from CStest


(b) Results from SegNet model trained only on V


(c) Results from SegNet model trained full cityscapes


(d) Results from SegNet model trained V+10CS


(e) Results from our MTL model

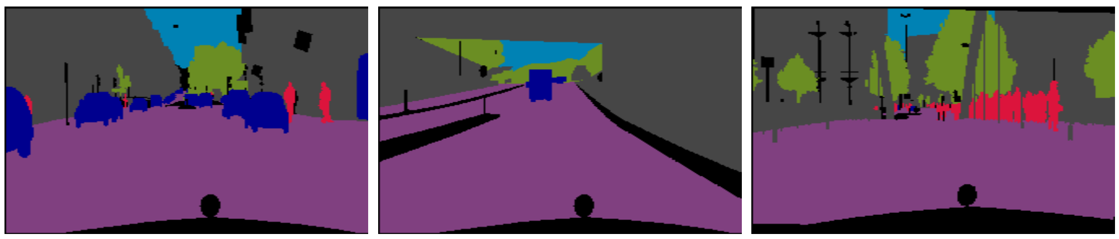
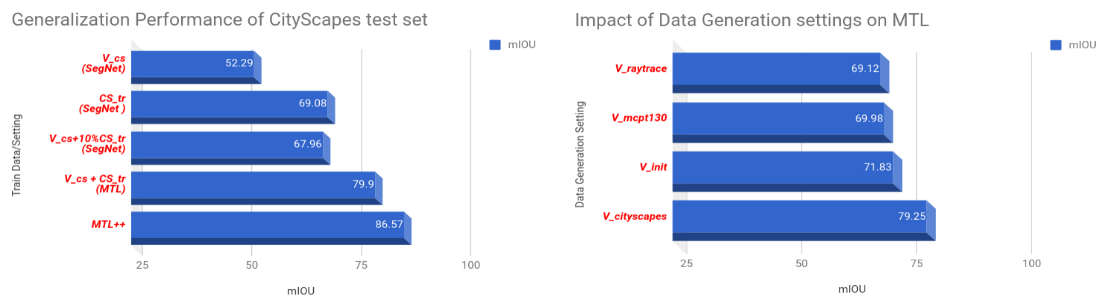

(f) Groundtruth labels

Figure 8.6: Semantic segmentation results



(a) Evaluation of generalization performance of MTL

(b) Impact of simulated data generation on MTL performance

Figure 8.7: Quantitative evaluations

dataset. Label prediction results of some samples are shown in Figure 8.6 for different training settings. Quantitative comparison of generalization performances on CS_val is provided in Figure 8.7.

8.3.1 Training only for Semantic Segmentation

We first use $V\_cityscapes$ to train SegNet and validate its generalization for CS_val. Please recall that $V\_cityscapes$ is the simulated data used in the previous chapter and that was generated by tuning our simulator to CityScapes training data (CS_train). As one can see in Figure 8.7a, the model trained only on $V\_cityscapes$ has shown a performance of 52.29 m-IOU points on CV_val. As already observed in the previous chapters, this model still suffers dataset bias issues, causing a performance degradation roughly by 17% compared to the model trained with full cityscape train split (CS_train). However, these domain-shift problems of simulated data can be mitigated by augmenting $V\_cityscapes$ with just 10% of CS_train. The model trained on such augmented data has reached a generalization performance levels (67.96 mIOU points) on par with that of the model trained on full CS_train (69.08 mIOU). Experimental protocol for these set of experiments is similar to the previous chapters.

8.3.2 Multi Task Learning

We now use our MTL framework to train SegNet on simulated datasets along with InpaintNet on unlabeled CityScapes. Please note that in this setting we use entire CS_train set but without its semantic labels. Generalization performance levels of the SegNet model are dramatically improved in all settings. This is nearly 27% (c.f. Figure 8.7a) improved when compared to the model trained only on $V\_cityscapes$. It also outperformed the model trained on full CS_train by 11%. More interestingly, the model achieved this accuracy without the need for any labeled real data but leveraging large-scale unlabeled data from CityScapes.

As already mentioned, we hypothesize that our MTL based architecture will force SegNet to learn geometric contextual features that are invariant to domain discrepancies. This means learned features should be generalizable to other datasets that are captured with a different camera or in different geographical regions. To empirically validate this hypothesis, we now evaluate the generalization capabilities of the above trained model (trained in MTL using *V_cityscapes* and *CS_train*) on test data from CamVid (*CV_val*). Our model achieves 87% performance levels is 10% better than what we achieved with full CityScapes. More interestingly, this is 8% more than what can be achieved with the full CamVid training set (*CV_train*). This provides empirical evidence supporting our claim that MTL encourages to learn features that are insensitive to appearance.

Since our MTL facilitates to use of unlabeled data, we make use of more data from CityScapes data which is unlabeled or partially labeled (a total of 20000 images). We also increased the size of labeled simulated data to 20000 samples. The SegNet model trained on this large scale dataset (denoted as MTL++ in Figure 8.7a) achieved 86.57 mIoU points which scores the state of the art performance on CityScapes.

8.3.3 Revisiting the impact of data generation settings

We now revisit our experiments in Chapter 5 and Chapter 7, and quantify the impact of scene generation and data rendering settings on the features learned in our MTL framework. First, to quantify the impact of scene generation parameters (c.f. Chapter 7), we used *V_init* in place of *V_cityscapes* to train our MTL architecture. As a result, we observed degradation in performance by 7%. Thus, we believe that tuning the parameters of scene generative model using real data from target domain should also be considered as a better practice towards efficient transfer learning from virtual to reality. However, the impact of rendering method becomes less significant since there is no major difference when we use the data rendered with ray tracer in place of *V_cityscapes* (rendered with Monte-Carlo path tracer).

8.4 Conclusions

In this chapter, we proposed an MTL-based architecture and training methodology to learn feature representations that are better transferable from virtual to reality. In this architecture, semantic segmentation network is trained on labeled simulated data while an auxiliary task of inpainting is trained using unlabeled real data. The results of our experiments proved that the feature representations that are learned in the proposed MTL framework are relatively less biased, thus, generalize well across the domains. Our MTL architecture can be extended in many directions such as learning spatiotemporal context using video-inpainting in temporal space or to cope with other auxiliary tasks such jigsaw-puzzle solving etc.

9 Conclusions

We conclude with a summary of the main contributions of the thesis and a consolidation of the experimental findings and observations. We also point out shortcomings of our tools and methods; and list some promising future research directions.

9.1 Summary

The first part of the thesis presented a complete pipeline for simulating labeled image datasets in large scale to train or diagnose modern vision systems such as deep convolutional networks. In the second part of the thesis, we addressed several fundamental issues about the impact of rendering choices and scene generation models on the generalization performance of DCNs trained on simulated data. In the third part, we design a novel MTL architecture that helps semantic segmentation network architecture to learn efficient feature representations that are well generalizable to the real-world target domain.

9.1.1 Tools

The details of our image rendering tool were explained in Chapter 3. Our tool was mainly implemented on top of BLENDER, an open source graphics engine. It is integrated with classical (such as Lambertian shading) and modern rendering engines (such as Monte-Carlo path tracing); and several annotation shaders. Thus, our tool can render the image data with a choice of rendering engine along with required annotations or ground-truth information. This tool is utilized to quantify the impact of modeling and computational approximations of the rendering engines on the generalization performance of the trained models.

Rendering images in large scale are required to synthesize several 3D scene states which include a detailed specification on 3D objects, light sources, camera and their intrinsic and extrinsic parameters etc. However, manually designing the 3D scene states is a time-consuming and laborious process. Hence, Chapter 4 proposed a stochastic scene generative model to automatically synthesize traffic scene layouts. Our generative model is based on Marked point processes coupled with pre-downloaded 3D CAD object shapes and factor potentials. The later chapters utilized these tools to address a set of scientific questions about the role of CG generated data in experimental vision.

9.1.2 Scientific experimentation and observations

Chapter 5 addressed the impact of rendering method and its parameters (modeling errors and computational approximations in rendering methods) on the generalization performance of a DCNN trained on simulated training sets. All the experiments were conducted in the context of traffic scene semantic segmentation with DeepLab [14] architecture. As expected, the virtual training data is required to have all photorealistic effects for better generalization of the trained models to real-world data. The *mIoU* performance of DeepLab on CityScapes testing set has been almost doubled when a raytracer (photorealistic rendering method, popularly used in video-games) is used for rendering in place of a basic classical Lambertian shader (that simulates only direct lighting effects). However, the physical accuracy of these photorealistic effects seemed to have a less significant impact on the generalization performance. Physics-based renderer MCPT has improved the results just by 6% over appearance-driven renderer ray tracing but at the cost of 15 times the rendering complexity. Furthermore, this improvement saturates around 40 Monte-Carlo samples-per-pixel. Hence, apparently extreme levels of photorealism may not be necessary. We also tried locating the image regions that are heavily biased when DeepLab is trained on virtual data. Our experiments revealed that the label estimates around object boundaries are heavily biased compared to other regions between virtual and real-world based training settings.

In Chapter 6, we proposed an unsupervised approach to tune the parameters of our scene generative model with a set of unlabeled samples from the target real domain. This was inspired by the recent advances in generative adversarial training that aims to train generative models by measuring the discrepancy between generated and real data in terms of their separability in the space of a deep discriminatively-trained classifier. Our method used an iterative estimation of the posterior density of prior distributions for our generative graphical model. Initially, we placed uniform distributions as priors on these parameters of a scene described by our generative graphical model. As iterations proceed the uniform prior distributions get updated to distributions that are closer to the (unknown) distributions of target data.

We used the proposed adversarial tuning approach in Chapter 7 to address the impact of parameter settings of the scene generative model on the generalization performance of DCNN. We have shown that tuning these parameters to the target domain will improve the generalizability of the simulated data on a testing set that comes from the target domain. Initial parameter setting used uniform distributions on scene parameters (lighting parameters, object sizes, orientations and weather scattering parameter, etc.) in their permissible ranges. The remaining two parameter settings were the distributions tuned to two existing real-world benchmarking datasets, i.e., CityScapes and CamVid. We demonstrated the utility of adversarially tuned scene generation on two real-world benchmark datasets (CityScapes and CamVid) for traffic scene semantic labeling with DeepLab. We realized performance improvements by 2.28 % and 3.14% between the DeepLab models trained on simulated sets prepared from the scene generation models before and after tuning to CityScapes and CamVid respectively.

Despite the improvements due to our tuning process, we still had to use a few labeled real samples (at least 9%) to correct the domain-shift problem. Hence, Chapter 8 presented a new multi-task learning based framework that uses "unlabeled" data from a target real-world domain to mitigate domain-shift issues of virtual data. This framework simultaneously trains two DCNs: one for semantic segmentation of simulated data and the other for an auxiliary self-supervised task on unlabeled data. Since our auxiliary task is supposed to learn the geometric context of the entire image, we choose to solve semantic inpainting problem. We also used an adversarial discriminator to help inpainting network produce a high-quality prediction. Our experiments prove that the MTL approach can learn feature representations that are well generalizable to real-world scenarios. In specific, MTL improves the generalization of SegNet by at least 11%. When in MTL, the impact of rendering engine has become insignificant, which means we don't have to use computationally intensive renderers such as MCPT. However, tuning of simulators still seems to important as data generated w/o tuning degraded the performance nearly by 7%. So, we think using scene models that are nearer to target domain and learning for multiple tasks are better practices when we are working with simulated data.

9.2 Future Research

Finally, we point out the shortcomings of our tools and approaches we are herewith practicing. However, we believe, despite these shortcomings, the tools and methodologies that we described will be of use to the CV community as a basis for pretraining, rigorous validation, and model refinement.

9.2.1 Stochastic Scene Models

For simplicity of simulation process, the base road network and ground plane underlying our generative scene model are fixed in geometry. Hence, our simulated images lack curved roads, terrains and complex road junctions. However, L-systems [66] or procedural generation techniques can be used to generate complex road networks. We used factor potentials to encode common-sense scene layout rules such as mutual exclusion, object-road relationships, etc. However, one can use advanced factor potentials to make probabilistic scene model more intelligent and informed with traffic constraints and rules that we find in real-world. We used a large collection of 3D object meshes and textures as a proxy to detailed probabilistic shape models for given object category. However, one can use parametric shape models from recent works of [57]. As most of the experimental setup has dealt with semantic segmentation on static images, we haven't focused much on motion and temporal context of the 3D scenes. Stochastic animation and path generation models can be used to extend our tools for automatic generation of scene dynamics and train video understanding systems such activity recognition and object tracking etc. Our probabilistic scene generator can also be used for likelihood-free inference (also referred as Approximate Bayesian Computation or Probabilistic Programming) for cognitive vision applications. The simulator in this framework should be able to generate several image samples from given scene priors, and stochastic comparator will accept or reject the parameters that have generated similar images compared a real input image to construct posterior [48, 42]. Utilizing our tools in the ABC frameworks is straightforward. However, inference will computationally intensive.

9.2.2 Learning in Simulators and Image refinement

Learning the parameters of generative models that could generate data statistically and visually similar to the given target unlabeled data is gaining recent attention [28]. Our scene generation model and its adversarial tuning procedure are first steps in this direction. As one future extension to our work, our simulator can be appended with a cascaded refinement procedure to transfer the appearance style of real images to simulated images by preserving the geometric content, thus, labels. This can be done using adversarial methods and can be trained in an end-to-end manner.

9.2.3 Learn to transfer spatiotemporal context

In chapter 8, we explore an auxiliary task of inpainting spatially. However, this can be extended to spatial-temporal space (volumetric inpainting) so that model will learn to transfer the spatiotemporal context from virtual to reality. We believe this would give more efficient transfer as temporal feature models also quite realistic in virtual worlds.

Bibliography

[1] https://threadreaderapp.com/thread/1031858472470495233.html.

[2] http://www.blender.org/.

[3] Anonymous. Adversarially tuned scene generation, 2017.

[4] Adrian J Baddeley and MNM Van Lieshout. Area-interaction point processes. *Annals of the Institute of Statistical Mathematics*, 47(4):601–619, 1995.

[5] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017.

[6] Jonathan Baxter. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning*, 28(1):7–39, 1997.

[7] Gabriel J Brostow, Julien Fauqueur, and Roberto Cipolla. Semantic object classes in video: A high-definition ground truth database. *Pattern Recognition Letters*, 30(2):88–97, 2009.

[8] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1872–1886, 2013.

[9] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *Computer Vision–ECCV 2012*, pages 611–625. Springer, 2012.

[10] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, volume 1, page 7, 2017.

[11] R Caruna. Multitask learning: A knowledge-based source of inductive bias. In *Machine Learning: Proceedings of the Tenth International Conference*, pages 41–48, 1993.

[12] Tony F Chan, Jianhong Shen, and Hao-Min Zhou. Total variation wavelet inpainting. *Journal of Mathematical imaging and Vision*, 25(1):107–125, 2006.

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

[14] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *arXiv preprint arXiv:1606.00915*, 2016.

[15] Robert L Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. In *ACM SIGGRAPH Computer Graphics*, volume 18, pages 137–145. ACM, 1984.

[16] 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. *arXiv preprint arXiv:1604.01685*, 2016.

[17] AJ Cox, Alan J DeWeerd, and Jennifer Linden. An experiment to measure mie and rayleigh total scattering cross sections. *American Journal of Physics*, 70(6):620–625, 2002.

[18] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[19] Carl Doersch, Abhinav Gupta, and Alexei A Efros. Unsupervised visual representation learning by context prediction. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1422–1430, 2015.

[20] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A learned representation for artistic style. *CoRR, abs/1610.07629*, 2(4):5, 2016.

[21] Long Duong, Trevor Cohn, Steven Bird, and Paul Cook. Low resource dependency parsing: Cross-lingual parameter sharing in a neural network parser. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, volume 2, pages 845–850, 2015.

[22] Mark Everingham, SM Ali Eslami, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, 111(1):98–136, 2015.

[23] Sanja Fidler, Sven Dickinson, and Raquel Urtasun. 3d object detection and viewpoint estimation with a deformable 3d cuboid model. In *NIPS*, 2012.

[24] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[25] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. *arXiv preprint arXiv:1605.06457*, 2016.

[26] Wenjie Ge and Robert T Collins. Marked point processes for crowd counting. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2913–2920. IEEE, 2009.

[27] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[28] 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.

[29] Michael Greiffenhagen, Visvanathan Ramesh, and Heinrich Niemann. The systematic design and analysis cycle of a vision system: A case study in video surveillance. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–704. IEEE, 2001.

[30] Peng Guan. *Virtual Human Bodies with Clothing and Hair: From Images to Animation*. PhD thesis, Brown University, Department of Computer Science, December 2012.

[31] Vladimir Haltakov, Christian Unger, and Slobodan Ilic. Framework for generation of synthetic ground truth data for driver assistance applications. In *German Conference on Pattern Recognition*, pages 323–332. Springer, 2013.

[32] Robert M Haralick. Performance characterization in computer vision. In *Computer Analysis of Images and Patterns*, pages 1–9. Springer, 1993.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[34] Geremy Heitz and Daphne Koller. Learning spatial context: Using stuff to find things. In *European conference on computer vision*, pages 30–43. Springer, 2008.

[35] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456, 2015.

[36] Martin Jacobsen. *Point process theory and applications.* Springer, 2006.

[37] Wojciech Jarosz. *Efficient monte carlo methods for light transport in scattering media.* ProQuest, 2008.

[38] Bela Julesz. Textons, the elements of texture perception, and their interactions. *Nature*, 290(5802):91, 1981.

[39] James T Kajiya. The rendering equation. In *ACM Siggraph Computer Graphics*, volume 20, pages 143–150. ACM, 1986.

[40] Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics (TOG)*, 31(4):55, 2012.

[41] Pushmeet Kohli, Philip HS Torr, et al. Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*, 82(3):302–324, 2009.

[42] Tejas D Kulkarni, Pushmeet Kohli, MSR Cambridge, Joshua B Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. *CVPR*, 2015.

[43] Florent Lafarge, Georgy Gimel'Farb, and Xavier Descombes. Geometric feature extraction by a multimarked point process. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(9):1597–1609, 2010.

[44] Hsin-Ying Lee, Jia-Bin Huang, Maneesh Singh, and Ming-Hsuan Yang. Unsupervised representation learning by sorting sequences. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 667–676. IEEE, 2017.

[45] Bastian Leibe, Konrad Schindler, Nico Cornelis, and Luc Van Gool. Coupled object detection and tracking from static cameras and moving vehicles. *IEEE transactions on pattern analysis and machine intelligence*, 30(10):1683–1698, 2008.

[46] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[47] Wallace Bishop Mann. Three dimensional object interpretation of monocular gray-scale images. 1996.

[48] Vikash Mansinghka, Tejas D Kulkarni, Yura N Perov, and Josh Tenenbaum. Approximate bayesian image interpretation using generative probabilistic graphics programs. In *Advances in Neural Information Processing Systems*, pages 1520–1528, 2013.

[49] Stephan Meister and Daniel Kondermann. Real versus realistically rendered scenes for optical flow evaluation. In *Electronic Media Technology (CEMT), 2011 14th ITG Conference on*, pages 1–6. IEEE, 2011.

[50] Rosana Montes Soldado and Carlos Ureña Almagro. An overview of brdf models. 2012.

[51] Srinivasa G Narasimhan and Shree K Nayar. Vision and the atmosphere. *International Journal of Computer Vision*, 48(3):233–254, 2002.

[52] Michael Oren and Shree K Nayar. Generalization of the lambertian model and implications for machine vision. *International Journal of Computer Vision*, 14(3):227–251, 1995.

[53] Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

[54] Adam Paszke, Soumith Chintala, Ronan Collobert, Koray Kavukcuoglu, Clement Farabet, Samy Bengio, Iain Melvin, Jason Weston, and Johnny Mariethoz. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration, may 2017.

[55] Deepak Pathak, Philipp Krahenbuhl, Jeff Donahue, Trevor Darrell, and Alexei A Efros. Context encoders: Feature learning by inpainting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2536–2544, 2016.

[56] Matt Pharr and Greg Humphreys. *Physically based rendering: From theory to implementation*. Morgan Kaufmann, 2004.

[57] Leonid Pishchulin, Stefanie Wuhrer, Thomas Helten, Christian Theobalt, and Bernt Schiele. Building statistical shape spaces for 3d human modeling. *Pattern Recognition*, 67:276–286, 2017.

[58] Ting Kei Pong, Paul Tseng, Shuiwang Ji, and Jieping Ye. Trace norm regularization: Reformulations, algorithms, and multi-task learning. *SIAM Journal on Optimization*, 20(6):3465–3489, 2010.

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

[60] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*, pages 759–766. ACM, 2007.

[61] Visvanathan Ramesh. *Performance characterization of image understanding algorithms*. PhD thesis, University of Washington, 1995.

[62] William T Reeves. Particle systemsa technique for modeling a class of fuzzy objects. *ACM Transactions on Graphics (TOG)*, 2(2):91–108, 1983.

[63] Stephan R Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. *arXiv preprint arXiv:1608.02192*, 2016.

[64] Cesar Roberto de Souza, Adrien Gaidon, Yohann Cabon, and Antonio Manuel Lopez. Procedural generation of videos to train deep action recognition networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[65] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3234–3243, 2016.

[66] Grzegorz Rozenberg and Arto Salomaa. *The mathematical theory of L systems*, volume 90. Academic press, 1980.

[67] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, abs/1706.05098, 2017.

[68] Alireza Shafaei, James J Little, and Mark Schmidt. Play and learn: Using video games to train computer vision models. *arXiv preprint arXiv:1608.01745*, 2016.

[69] Jamie Shotton, John Winn, Carsten Rother, and Antonio Criminisi. Textonboost: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *European conference on computer vision*, pages 1–15. Springer, 2006.

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

[71] Ruben M Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. A survey on procedural modelling for virtual worlds. In *Computer Graphics Forum*, volume 33, pages 31–50. Wiley Online Library, 2014.

[72] Charles Sutton, Andrew McCallum, et al. An introduction to conditional random fields. *Foundations and Trends® in Machine Learning*, 4(4):267–373, 2012.

[73] Olivier Tournaire, Nicolas Paparoditis, and Florent Lafarge. Rectangular road marking detection with marked point processes. In *Proc. conference on Photogrammetric Image Analysis*, 2007.

[74] Yanghai Tsin, Visvanathan Ramesh, and Takeo Kanade. Statistical calibration of ccd imaging process. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 480–487. IEEE, 2001.

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

[76] Akos Utasi and Csaba Benedek. A 3-d marked point process model for multi-view people detection. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3385–3392. IEEE, 2011.

[77] Tobi Vaudrey, Clemens Rabe, Reinhard Klette, and James Milburn. Differences between stereo and motion behaviour on synthetic and real-world stereo sequences. In *Image and Vision Computing New Zealand, 2008. IVCNZ 2008. 23rd International Conference*, pages 1–6. IEEE, 2008.

[78] David Vázquez, Antonio López, Daniel Ponsa, and Javier Marin. Cool world: domain adaptation of virtual and real worlds for human detection using active learning. In *Advances in Neural Information Processing Systems–Workshop on Domain Adaptation: Theory and Applications*, 2011.

[79] David Vázquez, Antonio M López, and Daniel Ponsa. Unsupervised domain adaptation of virtual and real worlds for pedestrian detection. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 3492–3495. IEEE, 2012.

[80] David Vazquez, Antonio Manuel Lopez, Javier Marin, Daniel Ponsa, and D Geroimo. Virtual and real world adaptation for pedestrian detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 36(4):797–809, 2014.

[81] Eric Veach and Leonidas J Guibas. Optimally combining sampling techniques for monte carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 419–428. ACM, 1995.

[82] Eric Veach and Leonidas J Guibas. Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76. ACM Press/Addison-Wesley Publishing Co., 1997.

[83] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shape modeling. In *Proc. CVPR, to appear*, volume 1, page 3, 2015.

[84] Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in neural information processing systems*, pages 341–349, 2012.

[85] Jiaolong Xu, Sebastian Ramos, David Vázquez, and Antonio Manuel López. Domain adaptation of deformable part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(12):2367–2380, 2014.

[86] Yongxin Yang and Timothy M Hospedales. Trace norm regularised deep multi-task learning. *arXiv preprint arXiv:1606.04038*, 2016.

[87] Lap Fai Yu, Sai Kit Yeung, Chi Keung Tang, Demetri Terzopoulos, Tony F Chan, and Stanley J Osher. Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (TOG)-Proceedings of ACM SIGGRAPH 2011, v. 30, no. 4, July 2011, article no. 86*, 2011.