University of Stuttgart
Institute for Signal Processing and System Theory
Professor Dr.-Ing. B. Yang

Masterarbeit D1391

GANs for LiDAR point cloud denoising

Author:	Bharath Bangalore Somashekar
Date of work begin:	20.01.2021
Date of submission:	20.07.2021
Supervisor:	Mr. George Basem Fouad Eskandar
Keywords:	LiDAR Point Cloud Denoising, GANs

LiDAR sensor is one of the most prominent sensors in an autonomous car to perceive the surroundings. The sensor maps the environment by sending light pulses. However, if the surrounding is affected by severe weather conditions, such as rain, fog, or snowfall, the light pulses are reflected from the weather particles and corrupt the LiDAR data. Hence, it is of paramount importance to identify, and remove the corrupted points and reconstruct the missing points in the LiDAR point cloud. In this aspect, some of the prominent deep learning architectures, such as CycleGANs, VAE, and WeatherNet are explored. Combining the noise awareness of WeatherNet and the point reconstruction of GANs, a novel method called MaskGAN is proposed. Furthermore, a new metric to measure the denoising and reconstruction performance of the model is also proposed. Additionally, LiDAR data processing using supervised learning methods, such as Pix2Pix on Resnet and UNet, is made possible by performing a pseudo pairing on the Cerema dataset. The experimental results show useful insights on the sparse point cloud generation capabilities of GANs and provide directions for improvement.

Contents

1. Introduction

Autonomous driving has been one of the most researched topics of the current times. The perception of the environment is made possible by the sensors connected to the car such as camera, radar, LiDAR, etc. A high-level understanding of the environment can be inferred by processing the data obtained from these sensors with certain algorithms. With the advancement in the field of computer vision, state-of-the-art methods can be used on the data captured from the camera to infer certain aspects of the environment. However, the data from the camera is specific to a field of view and it cannot be relied upon during times when the visibility of the environment is less. Hence, it is important to process the data from the other sensors and perceive the environment.

Light Detection and Ranging sensor, in short LiDAR, is one the most important sensors of the autonomous car, aiding perception by scanning the environment through light rays and inferring the distance of the object using the reflected light. LiDAR has a horizontal field of view of 360° and captures the complete surrounding of the car. Furthermore, LiDAR data captured is not prone to the varying visibility caused due to the change in time of the day and hence it plays a prominent role in the perception of the environment. The data obtained from the LiDAR sensor can be used for many inference tasks such as object detection, segmentation, scene prediction, and more.

A LiDAR transceiver emits rays of light to the surrounding environment, where, the light waves hit an object in the environment and the pulses reflect from the object back to the sensor. The sensor uses the time it takes the pulses to return back to calculate the distance of the object from the position of the sensor. A typical LiDAR sensor, mounted on top of an autonomous car, consists of a number of such LiDAR transceivers stacked vertically to form the scanning units of the sensor. The scanning units fire hundreds of pulses per second as the LiDAR sensor spins horizontally over the complete 360° angle to form a LiDAR scan of the surrounding. Each of the pulses returned can be processed into a three-dimensional visualization known as the point clouds through which the surrounding environment is depicted by the LiDAR sensor.

However, when the LiDAR sensor to used to record the environment prone to adverse weather conditions, such as severe snowfall, rainfall or even fog that reduces the visibility of the surrounding, the data recorded from the LiDAR sensor tends to be unreliable. During a LiDAR scan, the presence of weather particles in the environment, some of the light pulses unfavorably collide with the snow or the water particles and are bounced back to the sensor. These light pulses do not give information about the surroundings and hence are to be considered as outliers pulses. However, the sensor, along with the valid reflected pulses, considers these unfavorable reflections for the creation of point cloud. Hence, the created point cloud, depicting the environment, has points that are scattered by the weather particles and can be considered as corruption and in turn makes the perception of the environment unreliable. Hence, it is of paramount importance to pre-process the corrupted point cloud to remove the corruption before being processed from the algorithms for higher-level understanding of the environment.

The quintessential part of this master thesis is to identify the corruptions in the point clouds caused by weather particles, remove the corruptions from the point cloud and reconstruct the point cloud by generating points that would've been present if there was no scattering due to the weather particles by using deep learning paradigm. In this process, numerous LiDAR datasets are explored, LiDAR data representations are reviewed, existing metrics are discussed and new metrics are proposed and a number of deep learning networks are implemented and evaluated against performance to identify and remove the corruptions and reconstruct the LiDAR scan.

1.1. Motivation

Earlier methods of point clouds denoising use filters [1] [2] [3] to remove the noise from the laser scanned object. With the advent of deep learning methodologies, a plethora of supervised [4] and unsupervised [5] networks were proposed to remove the noise from the point cloud. These networks focus on removing the noise present in the dense point clouds containing. However, the point cloud in an autonomous car LiDAR scan is sparse and contains a lot more points than a point cloud created from scanning a small object and hence these methods cannot be used on point clouds based on LiDAR scan.

With respect to LiDAR point cloud denoising, there are specific filters [6] to remove the corruptions. However, in some of these filters, along with the corruptions, some of the points that are not affected by scattering are also removed making the LiDAR point cloud incomplete. Moreover, the point cloud is not reconstructed and hence always remains incomplete even if only the corruptions are removed.

There are very few deep learning networks that focus on removing the noise from the LiDAR point cloud such as the WeatherNet [7]. However, although this is a supervised network that can remove the corrupted points from the point cloud, it cannot reconstruct the point cloud and hence the LiDAR scan remains incomplete. In some of the works [8] [9], GAN [10], VAE [11] and cycleGAN [12], which is used for unsupervised image to image translation, is used to perform LiDAR translations and also to remove additive noise from the LiDAR. However, the corruptions caused by the weather particles result in scattering of the particles which in turn causes an unstructured noise in the LiDAR, which is more complicated and cannot be modeled by additive Gaussian noise. Furthermore, the generative networks used are ignorant of the noise and perform a simple image to image translation

Hence, there is a need for generative networks to encompass the noise awareness in a LiDAR point cloud to not only identify and remove the corruptions from the LiDAR scan but also to reconstruct the lost points. This thesis work embraces the shortcomings of the previous networks and encourages a noise aware reconstruction of the LiDAR point clouds

1.2. Contributions

The key contribution of the thesis work is as follows.

- Exploration and processing of various LiDAR datasets
- A range map based LiDAR point cloud representation used to process the data
- Review, implementation, and evaluation of the performance of some of the deep learning methods to remove the weather affected noise from the LiDAR point cloud
- Implementation of Dynamic radius removal filter (DROR)
- A novel mask based generative adversarial network called MaskGAN to incorporate the noise aware reconstruction of the LiDAR point clouds
- Novel metrics called the POP and RPOP to measure the denoising and the reconstruction performance of the network

1.3. Organization

This section provides a brief description of the rest of the chapters in the report.

Chapter 2 A discussion on the related works in point cloud denoising, LiDAR sparse point clouds denoising, and Generative networks in Point cloud processing is provided. Finally, the base methods used in this thesis work are presented.

Chapter 3 This contains a detailed discussion on the datasets available, various representations of the LiDAR data and the data processing on various datasets.

Chapter 4 The details of the implementation of various base methods are mentioned. A novel mask-based GAN is proposed and implemented. And supervised image-to-image translation approaches on various loss functions are explored.

Chapter 5 This entails the metrics used for evaluation and the quantitative and qualitative results are mentioned and discussed.

Chapter 6 This chapter provides a conclusion and the future work

2. Related Works

In this chapter, a general overview of point cloud denoising methods for dense and sparse point clouds is discussed, which is followed by a review of the existing adversarial networks for image generation and image to image translations. Followed by a discussion on point cloud processing using deep neural networks. In the end, some of the research that can be considered as base methods for the LiDAR point cloud processing are introduced.

Point Cloud Denoising Traditionally, there are many filters that are used to remove the noise from a 2D image [2] such as the bilateral filter [3, 13], non local means [14] or even sparse coding [15]. Some of these filters are also transferred to remove the noise from the point clouds. Based on the methods used, these filters can be divided into categories [16] such as statistical-based filters where it uses likelihood statistics [17] and other statistical methods to find out if a point is an outlier. Neighborhood-based filters use the neighboring points around a particular point to categorize the point. Similarly, projection-based filters such as locally optimal projection filter [18] uses different projection strategies to remove the outliers. Signal processing and the Partial differential equations (PDEs) based methods [16] uses the filters that correspond to certain methodologies. Furthermore, multiple filtering techniques can be combined [16] to remove the outliers or noise in the point clouds.

With the advent of deep neural networks and some of the earlier works such as PointNets [19] that worked directly on unordered point sets, some of the supervised learning models such as PointProNet [4], that can denoise the point clouds by projecting them to the learned local frame, PCPNet [20] is used to map the corrupted 3D point clouds to the clean point clouds, Neural projections [21], PointCleanNet [22] and similar networks demonstrated good performance on the supervised denoising task. Total denoising [5] is the first work on the unsupervised point cloud task using deep neural networks. This was further improved using differentiate manifolds [23] which produced state-of-the-art point cloud denoising performance

However, these methods would be effective on uniform point clouds containing a small number of points and would fail to perform well in a sparse point cloud setting, such as LiDAR point clouds.

LiDAR Point Cloud Denoising There are traditional filters that were used to remove the outliers from the sparse point clouds such as the LiDAR point cloud. Some of the prominent filters among the many are the SOR Filter, ROR filter, and Voxel grid filter. SOR filter [24], also called the Statistical outlier removal filter, calculates the mean distance between each point and its nearby points. The nearby points are inferred using K nearest neighbors. The sum of the mean distances and standard deviations are computed and if a point exceeds these numbers then it is considered an outlier and rejected. The performance of this filter depends on the number of neighbors considered and also the number of times the standard deviation is calculated. The disadvantage of this filter is that when the distance increases, the LiDAR point clouds become sparser, and many valid points get rejected. Also, finding the neighbors is computationally expensive and deteriorates the speed performance of the filter.

The ROR filter [24], also called the radius outlier removal filter, employs an algorithm to calculate the mean distance between each point and its neighbors within a given search radius using a k-d tree data structure. That point gets eliminated when the number of neighbors within a particular radius falls below a certain threshold. As a result, the filter's performance is determined by the radius and minimum point threshold that are given. One of the advantages of this filter is its ease of implementation considering a fixed search radius. Similar to the SOR filter, one of the downsides is that, as the distance of the Points increases, many points would be considered as outliers and removed.

Rather than using a distance-based approach, the voxel grid filter[24] uses a downsampling approach to eliminate the outliers. The 3D LiDAR point cloud is downscaled to a voxel center point that is later approximated. This filter not only removes the noise by downsampling but also some valid points based on the geometric information.

To overcome some of the drawbacks of the traditional LiDAR denoising methods, a few prominent modern works such as DROR filter [6] and the deep learning-based WeatherNet model demonstrate good performance in removing the noise from the LiDAR data. Also known as the Dynamic radius outlier removal filter, the DROR filter is an improvement over the traditional radius outlier filter. It addresses the issues mentioned in the traditional filters, where the traditional filters remove the valid points when the distance of the point clouds increases, by using a radius that is not fixed and changes when the distance varies. As a result, it enhances the accuracy of point clouds collected while driving in falling snow, and it is the most effective filter for eliminating snow and preserving the other elements in the field of view. However, the DROR was proposed and verified for removing the noise in the snowfall corrupted LiDAR point clouds and hasn't been verified for other weather corruption modalities such as rain or fog.

WeatherNet[7] is one of the very few deep learning-based models for LiDAR point cloud noise removal. This model is built on the Lilanet architecture [25] and used semantic segmentation to classify points as clear or fog or rain. This model not only uses the distance of the points to classify point clouds but also the intensity of the points. One of the main downsides of the models is that it requires supervised training wherein the semantic labels of the points classified have to be available at the training phase. A detailed overview of this method can be seen in the Base Methods and CNN Denoising

Generative Adversarial Networks (GANs) Generative Adversarial Networks (GANs) [10] are a form of adversarial networks that have become very famous due to their unfathomable capability in image generation tasks. It is being used in a plethora of applications such as image generation [26], image in-painting [27], image translations [12], image edition[28], and more. The basic GAN framework contains a generator and a discriminator. The generator and the discriminator of a Generative Adversarial Network (GAN) can be viewed as a two-player minimax game between two neural networks. The generator tries to figure out how to make data samples similar to real data samples, whereas, on the other hand, the discriminator is a binary classifier that tries to learn how to distinguish between real and generated fake samples. As this progresses, an equilibrium is reached where the generator can generate realistic samples and the discriminator can't identify what's real and what's not. The aim is to ensure that the distribution of the data generated is similar to the target data distribution. The generator G is specified as a map function, mapping the noise input z or the source data to the target data. The discriminator D is, on the other hand, a binary classifying network that discriminates between the real and the generated fake data. D and G are both trained at the same time. Discriminator D is trained to optimize the chance of classifying the data, either real or fake, with their right labels. The architecture is trained using an adversarial loss. With the plethora of applications comes a variety of modifications to the vanilla GAN.

Variational Autoencoder (VAEs) A VAE[11] is a simple extension of the traditional autoencoder[29] to incorporate probability distributions due to which there is a spectrum of applications of VAE [30] İt has been used in many image task such as [31] [32] [33]. Generally, in an autoencoder, the encoder can learn to map the input to a reduced-dimensional latent space. In latent space, the decoder uses the vector to generate the encoded image. A variational autoencoder (VAE) provides a probabilistic description of latent space observation. Therefore we formulate our encoder to describe a probability distribution for each latent attribute, rather than building an encoder that outputs a single value in the description of each latent state attribute.

Image to Image Translation In general, image to image translation is the conversion of a source image to a target image in which, the intrinsic source content is preserved and the extrinsic target style is added. The goal here is to find a mapping function that can map the source image to the target image to perform the conversion. Mathematically, this can be represented as shown, where x_B is the target image, x_A is the source image and $G_(A \rightarrow B)(x_A)$ is the mapping function $x_B = G_(A \rightarrow B)(x_A)$. I2I can be used in a broad range of applications, image segmentation [34], image in-painting [27], style transfer [35], 3D pose estimation [36], image colorization [37], cartoon generation [38], image super-resolution [39] and many more. Variational autoencoders [11] or the Generative Adversarial Networks [10] are the two widely used deep neural networks to make the image to image translation task feasible. VAEs, as described previously, have a wide range of applications in the I2I tasks, some works also suggest the training of the VAE is more stable than GANs [40]. However, VAEs sometimes fail to strike a good balance between the image reconstructed image and the source image [40]

Similar to VAE, GANs are widely used in a plethora of I2I tasks. The I2I methodology for GANs can be divided into two major categories. The supervised I2I and the unsupervised I2I. In the supervised I2I, there is an aligned pairing in the source and the target domain, which implicates that a target image for a particular source image is known while training the GANs. The GAN model will convert the source image to its corresponding target image. Pix2Pix [41] has been the front runner for the supervised image translation tasks which demonstrated state-of-the-art image to image translations and this inspired many other works that follow. Wang et al. [42] proposed DRPANs to overcome the blurry images produced by Pix2Pix, Pix2Pix HD [43] was introduced to produce higher resolution images. SelectionGAN [44] was introduced by Tang et al. to overcome the cross-view problem in translation. And similarly, BicycleGAN [45], SPADE [46], PixelNN [47], and more were introduced.

In the unsupervised setting, there are no aligned pairs between the source and the target domains. The model translates the source image to the target image by learning the source and the target distributions. In many cases, it is not possible to collect labeled data and hence unsupervised I2I task becomes a practical approach. CycleGAN [12] was the front-runner to use the cyclic consistency loss to perform unsupervised image to image translation. On the same lines, DualGAN [48], DiscoGAN [49] was introduced. UNIT [50] was proposed that made use of the shared latent space. TraVeLGAN [51] was proposed by Amodio et al. Benaim et al. introduced DistanceGAN [52] that maintains the distance between the source and the target domain. INIT [53], an instance aware GAN, was introduced by Shen et al., and similarly, more models were proposed overcoming the drawbacks of its predecessors.

Deep Learning for Point Clouds Deep learning methodologies on point clouds have been gaining a lot of attention in the recent. In the same lines, there has been numerous point cloud datasets such as ModelNet [54], ScanObjectNN [55], ShapeNet [56], S3DIS [57], PartNet [58], Semantic3D [59], ScanNet [60], KITTI Vision Benchmark [61], ApolloCar3D [62], Dense [63], Waymo Open Dataset [64], and many more available to the public. This corroborated the research in many of the tasks such as 3D shape classification, 3D object detection, object tracking, scene flow estimation, semantic segmentation, instance segmentation, point clouds completion and more.

Some of the deep learning models process the 3D point clouds as multi-view projections [65] and then use a Convolution neural network to perform tasks such as 3D shape classification, object detection, and more. MVCNN [66] is the front-runner which used max-pooling of multi-view features into a global descriptor. MHBN [67] used a bilinear pooling to combine the local convolutional features into a global descriptor. Many other methods have been introduced to increase the recognition accuracy, such as [68], [69]

Volumetric-based deep learning methods were widely used to process the point clouds data with a 3D convolution neural network [65]. A volumetric occupancy network called the VoxNet [70] was introduced by Maturana et al. to increase the robustness of the 3D object recognition. Octree-based 3D CNN was proposed by Wang et al. [71] Eventually, there came deep learning models that processed unordered point sets directly. PointNet [19] is a pioneering work in this field that used the 3D point clouds directly to perform classification tasks. Inspired by PointNet, many other works were introduced. PointNet++ [72] used the hierarchical networks to capture fine-grained geometrical details. To improve the point features, PointWeb [73] used the neighborhood information. Point Cloud GAN (PC GAN) was proposed to perform generation and reconstruction tasks in point clouds. Graph-based methods were also introduced that used each point as the vertex of a graph. Point Cloud GCN [74] is one such network that used graph convolution on the point clouds. Convolution-based methods performed 3D convolution on the point clouds to perform tasks such object detection, segmentation, and more. RS-CNN [75] takes some a subset of the overall points and performs a convolution. MCCNN [76] use monte carlo convolution on the point clouds.

There are advanced deep learning networks processing 2D images concerning image to image generation and denoising tasks and it is interesting to bridge the gap between the point clouds and the 2D images to use the advantages of the state-of-the-art 2D networks. In this line, some of the research, discussed in the next section, introduce a 2D representation of the 3D point clouds that can be processed with the state-of-the-art 2D deep learning networks and this is the methodology followed for the rest the models discussed in the next chapters.

2.1. Base Methods

There are some prominent literature that demonstrates the work on LiDAR processing, image translations, and also removing the noise in the LiDAR. We consider some of these works as the base methods to the task of removing the noise from the LiDAR point clouds

CNN-based LiDAR Point Cloud De-Noising in Adverse Weather This is the first CNN-based approach to remove the noise from the weather-affected LiDAR point clouds [7]. The noise consists of corruption from the weather, such as rain, or fog and this model is a significant leap from the state-of-the-art LiDAR denoising models. Along with this, some of the data augmentation techniques were introduced to induce rain and fog to the clear LiDAR data. The model is motivated by the argument that the existing LiDAR noise filtering algorithms work on the basis of discarding points based on the non-availability of neighboring points and hence some points would be wrongly discarded. Also, for instance, the rain droplets or the snowflakes would not just affect one single point but a cluster of it, and if one considers a neighbor-based filter then, the points might be wrongly classified and would affect the performance of operations carried on using the LiDAR sensor data. Also, usually, the intensity information of the points is not used in classifying the points, however, using the intensity of the points might provide better insight in classifying the points.

To overcome this use, the authors propose a segmentation-based LiDAR point denoising which used both the distance as well as the intensity information of the LiDAR scan in a range image representation as inputs and predicts the labels to classify each point as clear, rain, or fog points. To do this, an improved version of the Lilanet, called the WeatherNet, was developed.

The model was trained using the Cerema dataset [77], which was recorded in a climate chamber. The dataset also has the ground truth labels for all the points indicating if they belong to the rain, fog, or clear weather. Additionally, there is also a dataset whose LiDAR scans are recorded in the regular road conditions, and the rain and the fog augmentations are added to this dataset. The model was evaluated on many existing LiDAR denoising methods and was found to perform better on different conditions. Although the model is one of the pioneers in denoising the LiDAR data using a deep learning approach, the method requires the ground truth labels to be available for denoising. This would not be feasible in the real-world scenario.

Deep Generative Modeling of LiDAR Data [9] Sometimes the LiDAR scans would be corrupted or some of the data points would be missing, in this scenario, there is a need for the LiDAR scans to be reconstructed, and hence this work addresses this issue and focuses on the generation of point clouds and in turn synthesis of LiDAR scans. For this purpose, generative models such as Variational Autoencoders (VAEs) [11] and Generative adversarial networks [10] (GANs are used). For the data representation, the 3D point cloud is projected on a plane. The 2D projected maps can have 3 channels indicating the XYZ Cartesian coordinates and also a novel representation that used a polar distance channel and z channel to represent the point cloud. The authors claim that this representation yields better performance than the regular Cartesian coordinate representation of the 2D image.

The models were trained and evaluated on the KITTI dataset [61], since it is unsupervised learning, metrics such as EMD and CD are used to evaluate the performance of the models. The network performs well in removing the noise and also reconstructing the data points. This is one of the prominent models that used unsupervised methods to denoise and reconstruct the LiDAR scans. The method assumes a structured noise, that is drawn from the Gaussian distribution, however, in practical scenarios, the noise added to the LiDAR scans during the adverse weather, such as rain, fog, snowfall, and more, would be unstructured and cannot be represented in the Gaussian distribution. Also, the model uses no explicit noise-aware elements in its network and would not be completely suitable for a denoising application. We overcome this problem by introducing a noise aware element in our proposed network that is sensitive to the weather affected unstructured noise in the LiDAR scans

LiDAR Sensor modeling and Data augmentation with GANs for Autonomous driving [8] This work focuses on the translation of the LiDAR range images from simulation to real LiDAR images and also the upsampling of the LiDAR data. A sensor model is learned, that considers the realistic noise conditions such as imperfect reflections and material reflectivity when generating LiDAR data rather than the perfect simulated environment. To learn the sensor model, real data is used, nevertheless, it is not feasible to get a paired data containing simulated and real data for the same scenario, and hence this calls for the unsupervised learning of the sensor model. A CycleGAN is used to learn the sensor model, that can take the simulated LiDAR data and translate it to the real world noisy LiDAR data, where the noise is mainly from the material reflectivity and imperfect reflections. For the simulated LiDAR data, a recording of the Carla simulated LiDAR scans [78] is used. For the target data distribution, the KITTI [61] dataset is used. There are 2 types of data representations used in this work. Bird's Eye View (2D BEV) uses the LiDAR scans are projected onto the 2D grids. This would mainly be the top view of the LiDAR point clouds. However, the conversion to the point clouds from the BEV data is not possible as the elevation of the point clouds is lost in the BEV representation. On the other hand, the LiDAR point clouds are transformed into polar coordinates that exactly maps the LiDAR sensor's physical scan of the environment. This representation is called the Polar-Grid Map (2D PGM). Unlike BEVs, it is feasible to convert 2D PGM back to 3D PCL.

The CycleGAN was used to perform the tasks such as Sim2Real (BEV) and Sim2Real (PGM) to convert the simulated LiDAR scan to the real LiDAR scan also perform a Real2Real translation on BEVs to upsample a sparse LiDAR scan to a dense LiDAR scan.

A qualitative evaluation metric called the "annotation transfer" was proposed. This is evaluated by looking at the bounding boxes in the source image annotations and projecting them onto the generated images and verifying the regions with human visual assessments. These models perform well but their ability to denoise the LiDAR scans was not verified. In our implemented baselines we use the same CycleGAN to verify its capabilities to denoise the LiDAR scans

3. Data Processing

In this chapter, a general introduction to some of the prominent and publicly available LiDAR point cloud datasets is first discussed, additionally, a private simulated LiDAR dataset is also introduced. Gradually, the various data representations of the LiDAR point clouds are briefed and a detailed overview of the data processing on various datasets as well as the implemented conversion algorithms are discussed.

3.1. Datasets Available

Some of the popular datasets consisting of LiDAR data as discussed. However, the Seeing-Through-Fog and the CNN Denoising datasets are the only two datasets discussed that consist of the LiDAR data recorded in adverse weather conditions and hence are primarily used in this work.

3.1.1. KITTI Vision Benchmark

The KITTI Vision Benchmark dataset [61] comes from an autonomous driving platform and records six hours of driving with the help of digital cameras, LiDAR, and a GPS/IMU inertial navigation system. As a result, in addition to the LiDAR data, the imaging data is also provided. 7481 training photos and 7518 test images, as well as the accompanying point clouds, are included in the Object Detection Benchmark. Due to the moving scanning mode, the LiDAR data in this dataset is scarce. Cars, pedestrians, and bicycles are designated with bounding boxes.

KITTI LiDAR data, although widely used, does not contain any weather-related data and hence does not contain weather-induced noise.

The LiDAR points clouds are stored in the bin files and along with the XYZ Cartesian coordinate values, it consists of the intensity of the corresponding point cloud as well.

3.1.2. Waymo Open Dataset

The Waymo Open Dataset [64] is made up of high-resolution sensor data acquired by Waymo Driver-operated autonomous vehicles in a range of situations. This has 1 mid-range lidar that is on the top of the vehicle and 4 short-range lidars on the sides. There are 1150 scenes with high quality and synchronized recordings of LiDAR and camera data in wide geographical conditions consisting of urban and suburban regions.

The LiDAR at the top consists of 64 beams with a vertical FOV from $-17.6°$ to $2.4°$ and with dual returns, where, the strongest reflected light pulses are first received and recorded

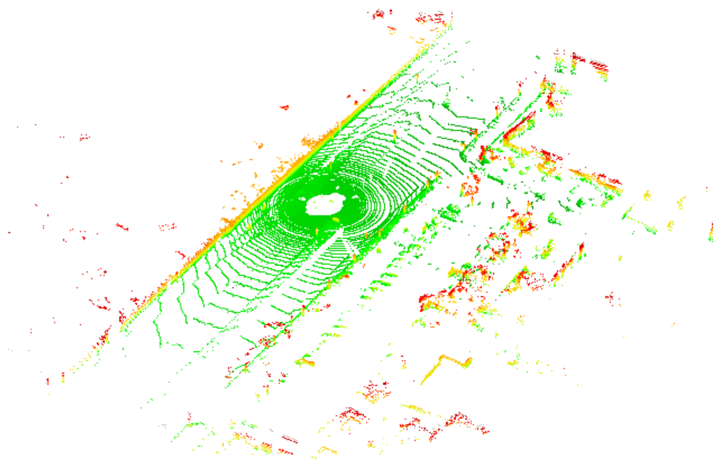


Figure 3.1.: LiDAR point cloud visualization of a sample scan from KITTI dataset

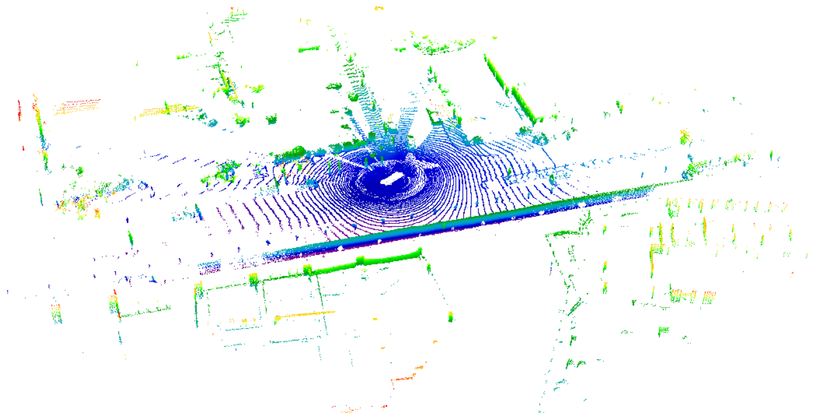


Figure 3.2.: LiDAR point cloud visualization of a sample scan from Waymo dataset

by the sensor, and later the second reflected light pulse with a lower intensity is received and processed.

The LiDAR data is by default available in Range Image format. However, this can also be converted to point clouds

Although Waymo is a big dataset consisting of diverse geographical conditions, similar to KITTI dataset, it does not contain any weather-related recordings.

3.1.3. Nuscenes Dataset

Nuscenes [79] by Motional is one of the largest collections of autonomous driving datasets containing a wide range of landscapes and other conditions. It has 7 times the annotations and 100 times the images as compared KITTI vision benchmark dataset. It has 6 cameras, 5 radars and a single LiDAR on the top of the driving car. LiDAR perceives the surroundings through which 1000 scenes through 390,000 LiDAR sweeps across Boston and Singapore were captured. This dataset contains some of the LiDAR sweeps that were captured in clear weather and rainy conditions and as well as in day and night times.

The LiDAR used has 32 beams, a 360° horizontal FOV and vertical FOV from $-30°$ to $10°$

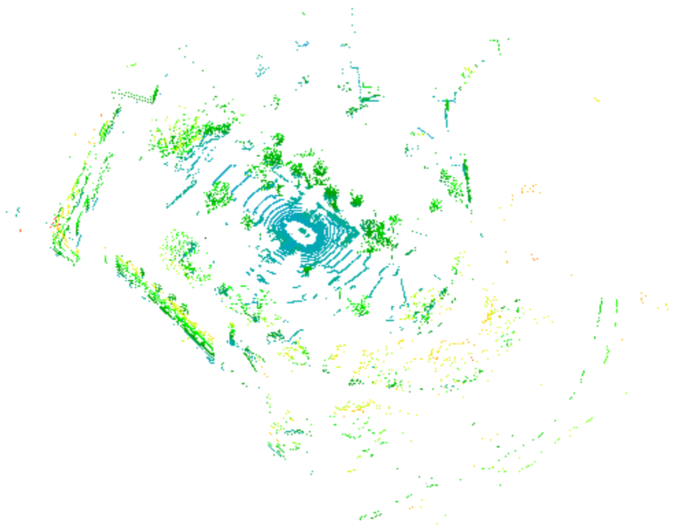


Figure 3.3.: LiDAR point cloud visualization of a sample scan from NuScenes dataset

The nuScenes LiDAR data is stored in bin format and contains the information of the Cartesian coordinates of the points as well as the intensity of the corresponding points.

NuScenes also provide LiDAR segmentation dataset called the nuScenes-lidarseg in which has annotations for each and every LiDAR point in the 40,000 key-frames of the nuScenes dataset

3.1.4. Carla LiDAR dataset

This is a private dataset generated by the Carla urban simulator [78]. The dataset consists of 2 LiDAR data. The first one is the 128 beams LiDAR data and the second one is the 32 beams LiDAR data. Both the LiDAR data consist of 1465 LiDAR scans each. The data is stored in the ply files and consists of Cartesian coordinates for each point in the point cloud. Unlike the other datasets mentioned, this dataset does not contain the intensity values for the points. A visualization of a sample LiDAR scan can be seen in the fig 3.4. As seen, the point cloud looks unnatural compared to the real LiDAR point clouds seen in the other datasets.

3.1.5. Seeing Through Fog

Similar to the previous datasets mentioned earlier, the Seeing through fog [63] dataset also contains the LiDAR scans from driving 10000 km, however, this is the first large multimodal dataset that primarily focuses on the sensor data, such as the camera, LiDAR, radar and, gated NIR, recorded on the car in a variety of adverse weather conditions at different times of the day driving around Europe.

The weather conditions captured in the dataset can be divided into clear weather, rain, fog, and snowfall. The LiDAR scan for clear weather conditions and weather-corrupted LiDAR scan can be seen in the figure 3.5. As seen, the fig a doesn't contain any corruption on the

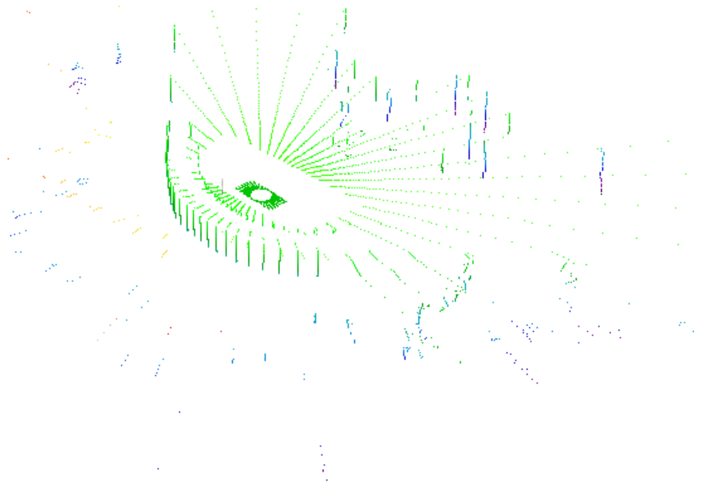
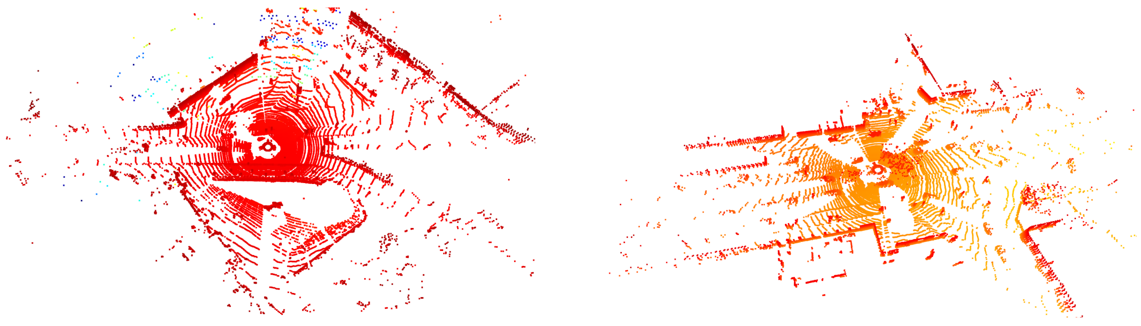


Figure 3.4.: LiDAR point cloud visualization of a sample scan from Carla dataset



(a) Clear weather LiDAR scan

(b) Snow corrupted LiDAR scan

Figure 3.5.: Seeing through fog LiDAR scans visualized for clear weather and snowfall conditions. As we can see in the second figure, the scattered points due to snow are accumulated in the vicinity of the LiDAR

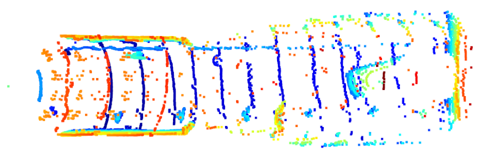
LiDAR scan as it is recorded in the clear weather, whereas, in the fig b, there is noise at the vicinity of the LiDAR transceiver due to ray scattering from the snow particles.

Similar to nuScenes and KITTI, LiDAR data is stored in bin format and contains the information of the Cartesian coordinates of the points as well as the intensity of the corresponding points. The labels corresponding to the weather conditions of the particular LiDAR sweep are also available.

There are 2 LiDARs from Velodyne on the top of the car. HDL64 S3D operates with 64 beams and VLP32C operates at 32 beams. They both dual return which consists of the strongest and the last returns. In the experiments performed in our current work, we use only the strongest return from 64 beams HDL64 S3D LiDAR device

3.1.6. CNN Denoising Dataset

This data present in this dataset can be divided into two types


(a) Clear weather LiDAR scan

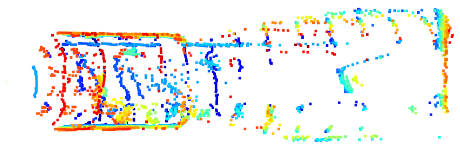

(b) Fog corrupted LiDAR scan

Figure 3.6.: Cerema climate chamber LiDAR scans visualized for clear weather and fog conditions. As we can see in the second figure, the scattered points due to fog are accumulated in the vicinity of the LiDAR

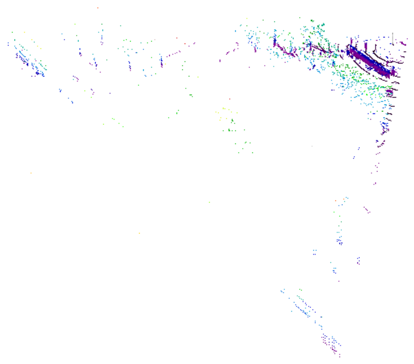


Figure 3.7.: LiDAR point cloud visualization of the road dataset

Cerema dataset: This contains only lidar data that was captured in the Cerema climate chamber with different simulated weather settings. The dataset 4 scenes out of different scenarios. The weather types can be enumerated as clear weather, fog, and rain. The fog can be further enumerated based on its visibility as fogA, fogB and fogC. Also, the rainy weather can be further enumerated based on its rate of rainfall. Each point of the lidar point clouds is labeled indicating different weather conditions. The figure 3.6 illustrates the clear weather LiDAR scan and the LiDAR scan corrupted for fog in the Cerema climate chamber.

A robosense LiDAR RS-LiDAR-32 containing 32 beams with the vertical FOV from 15° to −25° is used for recording the LiDAR data in the climate chamber

Road Dataset with weather augmentations: This dataset contains lidar data captured while driving on the roads. Noise related to the weather is induced on to this clear weather dataset. The weather augmentations used are as the Rain model and the Fog model [7]. The rain model induces more point scattering due to the higher presence of water droplets and also the intensity is affected. The fog model induces fog-related noise on the clear dataset with point scattering caused by water droplets and the intensity changes due to fog that would in real scenarios. The LiDAR point clouds from this dataset are sparse and a sample LiDAR scan can be seen in the figure 3.7

The models mentioned in the rest of the work would be trained on Seeing-Through-Fog or Cerema dataset as they contain weather-induced noises that are evident on the LiDAR point clouds.

3.2. Data Representations

3.2.1. Voxel Grid

Previously there were no deep learning methods that directly process the 3D point clouds and hence the point clouds were rasterized into voxel grids and then processed using 3D volumetric convolutions [80] [81]. This representation spawns many research works such as vanilla volumetric CNN prosed by Maturana et al [70], the 2D CNNs were extended to 3D and compared with multiview 2D CNNs by Qi et al [68]. To reduce the memory consumption of the voxel grids, Wang et.al [82] also incorporated octree into volumetric CNNs. Also, many works used this representation for 3D object detection [83] and shape segmentation [84]

This representation needed a high large memory footprint to save the data in voxels. Reducing the number of voxels would result in the fine-grained details being lost and hence high-resolution data is to be stored, which would increase the computational cost and the memory footprint three folds with the increase in the voxel resolutions.

3.2.2. Unordered Point Sets

A point cloud is a set of n points on a 3D space $P = \{p_i \mid i = 1, \ldots, N\}$, where each point $p_i \in P$ can be represented as a vector (X, Y, Z) where X, Y, Z represents the Cartesian coordinates of the point. In recent years, many works have focused on directly processing the point clouds rather than converting them to any representations. PointNet [19] was one such work that laid the foundation for many other networks that can use the point clouds directly. This architecture makes use of the symmetric functions of the point clouds to process them in 3D, gradually many other works proposed to stack PointNets hierarchically [72] to extract neighborhood information. Since then, many models have been developed that demonstrate state-of-the-art performance in 3D object detection, 3D semantic segmentation, and more. Compared to Voxel grids, the point-based models are less GPU memory intensive as the memory needed is to only capture the point coordinated in 3D space.

The set of points in the 3D point cloud are invariant to $N!$ permutations, in other words, there is no specific order. They are also invariant to rotation and translation and always maintain the same geometric structures. The neighboring points on the 3D space, separated by some distance, can result in meaningful information regarding the local structures of the point cloud.

However, owing to the fact that the points are scattered in 3D space, unlike voxel grids, accessing a point in random memory is inefficient. The neighbors a point is not stored in contiguous memory locations and hence accessing the neighbors to extract the information of the local structures penalize the network and hence it is one of the disadvantages of using the unordered point sets representation. One of the recent works, Point-Voxel CNN [85], took advantage of both the representations by combining them as inputs, whereby the model can be aware of the local structures as well is fast enough to access random memory. However, a unified representation that can cater to the needs of memory, and neighborhood information is needed.

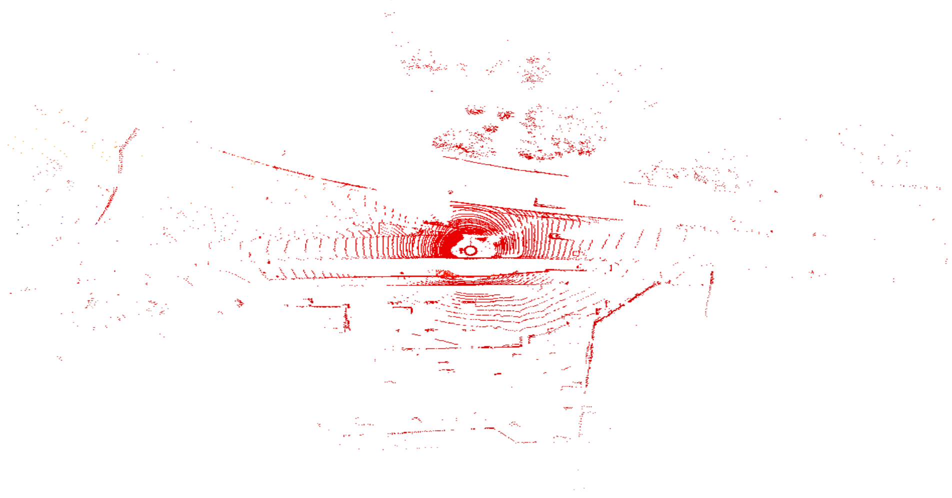


Figure 3.8.: Bird eye view of one of the Seeing-Through-Fog LiDAR scan

3.2.3. Bird Eye View

In this representation, the 3D point cloud is projected onto a 2D plane. As the name suggests, the 2D plane is the view that is seen from the top of the 3D point clouds. BEV representation has been used in many works and has proved effective in tasks such as 3D object detection [86], road detection [87] as well as semantic segmentation [88]. Furthermore, in some of the works performing unsupervised LiDAR to LiDAR translations [8], BEV representation of the LiDAR data was used extensively.

Although the representation is simple and less processor and memory intensive, once the 3D LiDAR point clouds are mapped to BEV representations, they cannot be mapped back to point clouds as the rest of the points occluded by the top view are lost and hence this is one of the biggest disadvantages of this representation

The figure 3.8 illustrate the bird view of a LiDAR scan from the Seeing-Through-Fog dataset

3.2.4. Range map

The 3D point clouds are converted to the 2D points by making use of the polar or spherical coordinates. For polar coordinates, the point $P = (x, y, z)$ can be mapped to a 2D cylindrical plane $Q = (\theta, \varphi)$, where θ is the azimuth angle and φ is the elevation angle from the origin. The points are arranged in a matrix of size HxW, where H indicates the elevation axis and W indicates the azimuth axis. The range of θ and φ depends upon the LiDAR sensor configurations where most of the points would be captured. The value of each cell would be the euclidean distance of a point from the origin on a horizontal 2D plane, i.e $d = \sqrt{x^2 + y^2}$. For cells that don't contain any points, the value would be equated to zero.

For a spherical coordinate, the point $P = (x, y, z)$ can be represented on a 2D spherical plane $S = (\theta, \varphi, r)$, where θ and φ are the same as in the polar coordinates but r is the euclidean distance of the point from the origin, i.e $r = \sqrt{x^2 + y^2 + z^2}$. The value of r will be the value of each cell on the HxW matrix which is similar to the polar coordinates.

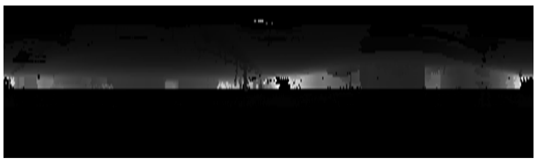


Figure 3.9.: Range Image of one of the Seeing-Through-Fog LiDAR scan

There are other variants to this form of representation, one such variant is introduced in [9] where a polar coordinate is used, and along with the cylindrical distance, one more channel is added which contains the elevation of the points. Similarly, there are other forms that contain 3 channels indicating 3 axes in Cartesian coordinates. All the forms of range map representations mentioned above can be mapped back to 3D point clouds but with some losses compared to the original 3D point clouds.

The range map representations are usually prone to quantization error, where the size of the matrix, i,e. H and W decide the quantization error of representing the 3D point clouds on a 2D plane. Higher the values of H and W, then more points are uniquely represented and the quantization error reduces but this makes the size of the matrix bigger and sparser, and hence the value of H and W is chosen as a trade-off between size and the quantization error. In some of the works containing range maps, the value of a cell in the HxW matrix would be the average distances of all the points falling to the same cell. This form, however, will make the overall quantized 3D point cloud look unnatural. Motivated by the concept of occlusion, where the environment behind an object is unknown, we introduce a new form of selecting the value of the cell by choosing the point which has the lowest distance of all the points falling in the same cell.

For the Seeing through fog dataset, the θ is chosen to be between 0 to 360° and φ to be between -30 to 30°, the horizontal angular resolution will be $360/W$ and the vertical angular resolution will be $60/H$. The H and W are chosen to be 32 and 400, indicating 32 channels LiDAR with 400 ray casting by each channel in each LiDAR sweep.

For the Cerema dataset which uses a Robosense LIDAR unit, the dataset directly provides the points in a 2D range map representation containing spherical coordinates of size $32x400$. The main challenge here is to map the points back to their 3D point cloud representation. For this, θ is chosen to be such that it starts with a value of 40° and gradually decreases to $-40°$ with a horizontal angular resolution of 0.20° indicating 400 cells of each row in the matrix. The elevation angle φ starts from 15° and move to $-24°$ in a non-uniform vertical angular resolution ranging from 0.33 to 4.66° indicating the 32 channels of the LiDAR.

The range map can be mapped back to the 3D point clouds by finding the coordinates representing each value in the range map using the distance, elevation, and azimuth angles. In case of spherical coordinates, $z = r\sin(\varphi)$, $d = r\cos(\varphi)$, $x = d\cos(\theta)$, $y = d\sin(\theta)$ where r is the spherical distance and d is the cylindrical distance

The figure 3.9 illustrate the range image of a LiDAR scan from the Seeing-Through-Fog dataset

3.3. Data Processing

3.3.1. Converting point cloud into range image in Cartesian coordinates

The main idea is to convert the point clouds into range images with Cartesian coordinates. Usually, in most of the datasets, the data available will be in the form of Cartesian coordinates - x,y,z points indicating the point in the free space. For the purpose models in this paper, converting these point clouds coordinates to the range image of configurable height and width is important. A theoretical understanding of the process of converting the point clouds to the range image is explained in the previous section, the code snippet for the generic conversion to range image can be seen in 3.1. The function takes in the point coordinates, the minimum and the maximum vertical angle of the LiDAR used, and the required height and width of the range image. By default, the horizontal scan angle is considered to be from 0 to 360°. As mentioned previously, if multiple points are present in the same cell, then the point that has the minimum distance from the origin is considered. In this code snipped mentioned, the intensity of the points is also passed along with the xyz points and the function, along with the coordinates range image, returns the compact intensity range image of the corresponding points in the range map. This function returns the range image with a depth of 3 indicating the 3 coordinate values xyz. This can be converted to the spherical distance or the polar distance as per the need. In the models used in this work, mostly spherical distances are used in the range images. The function to convert to the spherical values can be seen in the code snipped 3.2.

Listing 3.1: Generic Range Image Conversion

```python
import numpy as np

def pointcloud_to_depth_map(pointcloud: np.ndarray, theta_res=400,
    phi_res=32, phi_min_degrees=60,
                        phi_max_degrees=120) -> np.ndarray:

    assert pointcloud.shape[1] == 3, 'Must have (N, 3) shape'
    assert len(pointcloud.shape) == 2, 'Must have (N, 3) shape'

    xs = pointcloud[:, 0]
    ys = pointcloud[:, 1]
    zs = pointcloud[:, 2]
    ins = pointcloud[:, 3]

    rs = np.sqrt(np.square(xs) + np.square(ys) + np.square(zs))
    phi_min = np.deg2rad(phi_min_degrees)
    phi_max = np.deg2rad(phi_max_degrees)
    phi_range = phi_max - phi_min
    phis = np.pi/2 + np.arcsin(zs / rs)

    THETA_MIN = -np.pi
    THETA_MAX = np.pi
    THETA_RANGE = THETA_MAX - THETA_MIN
    thetas = np.arctan2(ys, xs)
    thetas_in_Degree = np.rad2deg(thetas).tolist()

    phi_indices = ((phis - phi_min) / phi_range) * (phi_res - 1)
    phi_indices = np.rint(phi_indices).astype(np.int16)
    theta_indices = ((thetas - THETA_MIN) / THETA_RANGE) * theta_res
    theta_indices = np.rint(theta_indices).astype(np.int16)
    theta_indices[theta_indices == theta_res] = 0

    canvas = np.zeros(shape=(phi_res, theta_res, 4), dtype=np.float32)
    intensity = np.zeros(shape=(phi_res, theta_res, 1), dtype=np.float32)

    valid_phi_indices = []
    valid_theta_indices = []

    for i in range(phi_indices.shape[0]):
       if phi_indices[i] >= 64 or phi_indices[i] < 0:
          continue
       else:
          valid_phi_indices.append(phi_indices[i])
          valid_theta_indices.append(theta_indices[i])

          if canvas[phi_indices[i],theta_indices[i],0] == 0 or
             canvas[phi_indices[i],theta_indices[i],0] > xs[i]:
             canvas[phi_indices[i],theta_indices[i],0] = xs[i]

          if canvas[phi_indices[i],theta_indices[i],1] == 0 or
             canvas[phi_indices[i],theta_indices[i],1] > ys[i]:
             canvas[phi_indices[i],theta_indices[i],1] = ys[i]

          if canvas[phi_indices[i],theta_indices[i],2] == 0 or
             canvas[phi_indices[i],theta_indices[i],2] > zs[i]:
             canvas[phi_indices[i],theta_indices[i],2] = zs[i]

          if intensity[phi_indices[i],theta_indices[i],0] == 0:
             intensity[phi_indices[i],theta_indices[i],0] = ins[i]

    phi_indices = np.asarray(valid_phi_indices).reshape(-1,1)
    theta_indices = np.asarray(valid_theta_indices).reshape(-1,1)


    return canvas[:,:,:3], intensity
```

Listing 3.2: Conversion to spherical distance from the cartesian coordinates

```
def to_spherical_np(pcd):
    if len(pcd.shape) == 3:
        dist = np.sqrt(pcd[:, :, 0] ** 2 + pcd[:, :, 1] ** 2 + pcd[:, :, 2]
            ** 2)
        dist = np.expand_dims(dist, axis=2)
    elif len(pcd.shape) == 2:
        dist = np.sqrt(pcd[:, 0] ** 2 + pcd[:, 1] ** 2 + pcd[:, 2] ** 2)
        dist = np.expand_dims(dist, axis=2)
    else:
        dist = None

    return dist
```

3.3.2. Converting from the range image to point cloud

As explained in the previous section, the conversion of range images to the point clouds depends upon the LiDAR configurations. The main idea here is to figure out the x, y, and z coordinates of a point considering its azimuth and the horizontal angle along with its spherical distance value. The function takes the LiDAR configurations are parameters such as the minimum and the maximum vertical angles and the height and width of the range images. There are variants for the code that takes in range image with spherical distances or range image with polar distances along with the z value. This can be seen in the code snippets 3.3.

Listing 3.3: Conversion to point clouds from generic range images

```
def from_spherical(dist, phi_min=60, phi_max=120, row=32, col=400):

    dist = torch.squeeze(dist, 3).cuda()

    phi_max = np.deg2rad(phi_max)
    phi_min = np.deg2rad(phi_min)
    phi_res = (phi_max-phi_min)/row
    init_phi = phi_min - np.pi/2

    theta_init = -np.pi
    theta_Res = 2*np.pi/col

    temp_torch = torch.linspace(0, (row*col - 1), steps=row*col,
        dtype=torch.int32).reshape(row,col).cuda()
    row_tensor = torch.div(temp_torch, col).int().cuda()
    phi_tensor = torch.add(torch.mul(row_tensor,phi_res), init_phi).cuda()

    col_tensor = torch.remainder(temp_torch, col).int().cuda()
    theta_tensor = torch.add(torch.mul(col_tensor,theta_Res),
        theta_init).cuda()

    phi_tensor = torch.unsqueeze(phi_tensor, 0).cuda()
    theta_tensor = torch.unsqueeze(theta_tensor, 0).cuda()
    batch = list(dist.shape)[0]
    batch_phi_tensor = phi_tensor.repeat(batch, 1,1).cuda()
    batch_theta_tensor = theta_tensor.repeat(batch, 1,1).cuda()

    z = torch.mul(torch.sin(batch_phi_tensor).cuda(), dist).cuda()
    d2d = torch.mul(torch.cos(batch_phi_tensor).cuda(), dist).cuda()
    y = torch.mul(torch.sin(batch_theta_tensor).cuda(), d2d).cuda()
    x = torch.mul(torch.cos(batch_theta_tensor).cuda(), d2d).cuda()

    out = torch.stack([x,y,z], dim=3)

    return out
```

3.3.3. Data Pre-processing

Data loaders Data loaders that can load the range image data in Cartesian, polar, or spherical coordinates are implemented for all the mentioned datasets. The data loader takes care of the necessary data split - train, validation, and test - and also in datasets like Cerema and Seeing-Through-Fog, it takes care of data loading of the LiDAR scan in the mentioned weather condition. By default, it will return the range image data in Cartesian coordinates, this can be controlled by passing transformation methods such as $to_spherical$ or to_polar during creation. Data loaders pertaining to Cerema and Seeing-Through-Fog return the intensity information along with the coordinates range image.

Normalization The range images before being returned are normalized to have the values between 0 and 1 by dividing the elements with the maximum value available in the corresponding range image. This holds true in the case of the intensity information as well where the intensity values are divided by the maximum intensity values.

Pseudo Pairing A supervised network would need aligned source and target data to perform an image translation however, with respect to the LiDAR point cloud denoising, it is hard to find paired corrupted and clear point cloud data. In the Cerema dataset, although no paired data is available, the clear weather lidar scan and the corrupted lidar scans for each weather condition can be found for the same scenes. And hence, instead of pairing the exact corrupted data to the corresponding clear weather data, a pseudo pairing is used where each corrupted data to paired to a random clear weather data from the scene. This is used to train and test the supervised models mentioned in the later sections

Special Cases in Cerema Dataset The Cerema dataset can be considered as a special case where the data is immediately available as a range image with spherical distances. The LiDAR used is the Robosense 32 which has a vertical resolution that varies from 0.33 to 4.66°, a vertical angle ranging from +15 to −25°, and horizontal resolution as 0.09, 0.18, or 0.36° with a complete 360° scan. However, after reverse-engineering the Cerema range image, it was found that the horizontal angle starts from 39.9° and increments with a horizontal resolution of 0.20°, also the vertical angle resolution is nonuniform. Hence, a modified function was implemented to convert the Cerema range images to point clouds. The code snippet can be seen in 3.4

Listing 3.4: Conversion to point clouds from the Cerema range images

```
angle_list = [15.0, 10.33, 6.99, 4.66, 3.33, 2.33, 1.66, 1.33, 0.99, 0.66,
    0.33, 0.0, -0.33, -0.66, -0.99, -1.33, -1.667, -1.99, -2.33, -2.66,
    -3.0, -3.33, -3.66, -3.99, -4.66, -5.33, -6.14, -7.25, -8.84, -11.30,
    -15.63, -24.99]

def from_spherical_cerema(dist, row=32, col=400):

    angle_tensor = torch.Tensor(np.deg2rad(angle_list)).cuda()
    angle_tensor = angle_tensor.reshape(-1,1)
    dist = torch.squeeze(dist, 3).cuda()
    phi_max = np.deg2rad(phi_max)
    phi_min = np.deg2rad(phi_min)
    phi_res = np.deg2rad(4.66)
    theta_init = np.deg2rad(39.9)
    theta_Res = np.deg2rad(0.20)

    temp_torch = torch.linspace(0, (row*col - 1), steps=row*col,
        dtype=torch.int32).reshape(row,col).cuda()
    row_tensor = torch.div(temp_torch, col).int().cuda()
    phi_tensor = angle_tensor
    col_tensor = torch.remainder(temp_torch, col).int().cuda()
    theta_tensor = torch.sub(theta_init,
        torch.mul(col_tensor,theta_Res)).cuda()
    phi_tensor = torch.unsqueeze(phi_tensor, 0).cuda()
    theta_tensor = torch.unsqueeze(theta_tensor, 0).cuda()

    batch = list(dist.shape)[0]
    batch_phi_tensor = phi_tensor.repeat(batch, 1,1).cuda()
    batch_theta_tensor = theta_tensor.repeat(batch, 1,1).cuda()

    z = torch.mul(torch.sin(batch_phi_tensor).cuda(), dist).cuda()
    d2d = torch.mul(torch.cos(batch_phi_tensor).cuda(), dist).cuda()
    y = torch.mul(torch.sin(batch_theta_tensor).cuda(), d2d).cuda()
    x = torch.mul(torch.cos(batch_theta_tensor).cuda(), d2d).cuda()
    out = torch.stack([x,y,z], dim=3).cuda()

    return out
```

Also, with respect to the pre-processing, there are some points in the Cerema LiDAR scan that usually are with very high distances. These are mostly outliers that posed problems for

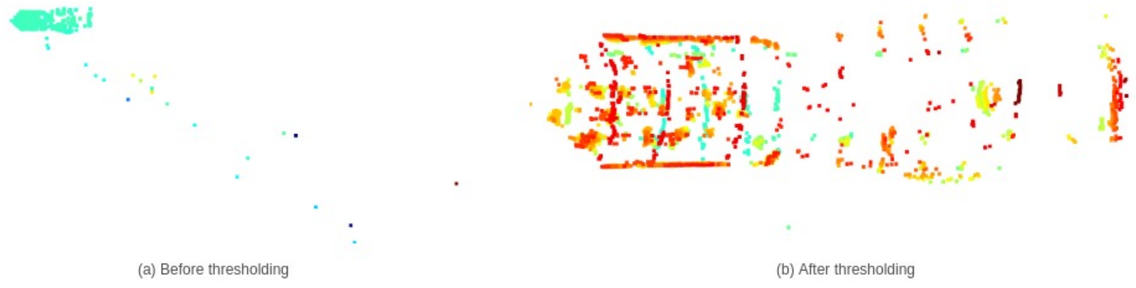


Figure 3.10.: Cerema dataset distance thresholding to remove outliers

the normalization. Statistically, it was found most of the main scans lie within the distance of 28 meters, and hence the dataset was pre-processed to consider only the points that are below this threshold and later normalized to be between 0 and 1. Figure 3.10 illustrates the problem. The point cloud on the left side is with outliers and hence the main scan becomes miniaturized because of the outliers, this is overcome by applying a threshold as shown in the point cloud in the right.

Seeing-Through-Fog weather segregation Originally, in the STF dataset, the labels regarding the lidar scans are present in a JSON file that indicates to which weather condition does a particular scan belongs. This file was processed in such a way that for each weather condition a new text file was created that consists of the path of that particular LiDAR scan. The data loader uses the text file for the corresponding weather requested. Additionally, a text file was created that contained all the paths for the LiDAR scans that were corrupted due to any condition. This can be generically used by the data loader to access the corrupted LiDAR scan.

3.3.4. Data split

Seeing-Through-Fog and the Cerema datasets are the ones that are mainly used in the models mentioned in the next chapters and hence it is necessary to discuss the data splits for training, testing, and validation for these datasets.

Seeing-Through-Fog Dataset There is no implicit data split available in the dataset, so as mentioned earlier, the whole of the data is segregated into different weather conditions and the split is applied on this segregated data while data loading. 60 percent of the data in the weather conditions considered is taken to be the training set, 20 percent of the data is considered to be in the validation set and the remaining 20 percent is the test set.

Cerema Dataset There is implicit data split in the dataset and this is the split used in the models seen in the next chapters. Inherently, the dataset consists of the recording of 4 different scenes in both day and night conditions for each scene and each weather. The training set consists of 2 scenes, the validation set, and the test set consists of recordings from a single scene each.

Table 3.1.: Mean noise ratio in Cerema dataset

Scene	FogA	FogB	FogC	Rain15	Rain33	Rain55
train-01-day	-	0.052	0.024	0.325	0.022	0.075
train-01-night	0.109	0.12	-	-	0.02	0.089
train-02-day	-	-	-	0.32	0.018	0.089
train-02-night	0.15	0.079	0.146	0.329	0.021	0.104
val-01-day	0.07	-	-	0.33	0.02	0.098
val-01-night	0.086	0.096	-	0.30	0.028	0.10
test-01-day	-	-	0.032	0.33	0.022	0.084
test-01-night	0.10	0.079	0.071	0.33	0.030	0.103

Additionally, the average of the ratio of noise to the number of points in each weather category of all the four scenes can be seen in the table 3.1

4. Methods

4.1. Base Models Implementation

4.1.1. CycleGAN

CycleGAN has proved useful in many of the image translation tasks, one of the main advantages being that it is unsupervised and learns the mapping between source distribution and the target distribution instead of requiring paired data. This is advantageous when it comes to LiDAR range map data translations as it's very difficult to get paired LiDAR datasets. Hence, we use this model in our work as one of the base models to evaluate the performance of the model on our datasets.

CycleGAN introduces the concept of cyclic consistency in GAN, through which it achieves unsupervised translations. Given two distributions, N and C, the objective is to adopt the CycleGAN to find the mapping between $n \sim P_d(n)$ to $c \sim P_d(c)$ which in turn maps the noisy data N to clear data C. The mapping in CycleGAN is achieved through two generator G and F, that try to map between N $\rightarrow$ C and C $\rightarrow$ N respectively and two discriminators D_x and D_y. The generator G will map from the noisy LiDAR data distribution N to clear weather LiDAR data distribution C while the discriminator D_x tries to differentiate between the real clear weather data C, and the generator G generated data, G($N \rightarrow C$). Similarly, the reverse generator F will map the clear LiDAR data C to the noisy LiDAR data distribution N while the discriminator D_y tries to differentiate between the real noisy data and the noisy data generated by F($C \rightarrow N$). The overall CycleGAN workflow is depicted in the figure 4.1

The architecture of the generator is depicted in the figure 4.2. Both of the generators used in CycleGAN are of the same architecture. Briefly, the generator architecture can be categorized as follows:

Generator Architecture:

- Initial Convolution block: This is the first block of the generator and it contains the reflection pad whose output would be forwarded to a convolution layer followed by instance normalization and a relu.
- Downsampling block: The output from the previous block is downsampled using a combination of convolution layer followed by instance normalization and relu activation layer.
- Residual block: This contains a repeated conv2D, instance norm and relu block and processes the downsampled data
- Upsampling block: The output from the final residual block is upsampled using a transpose convolution layer.

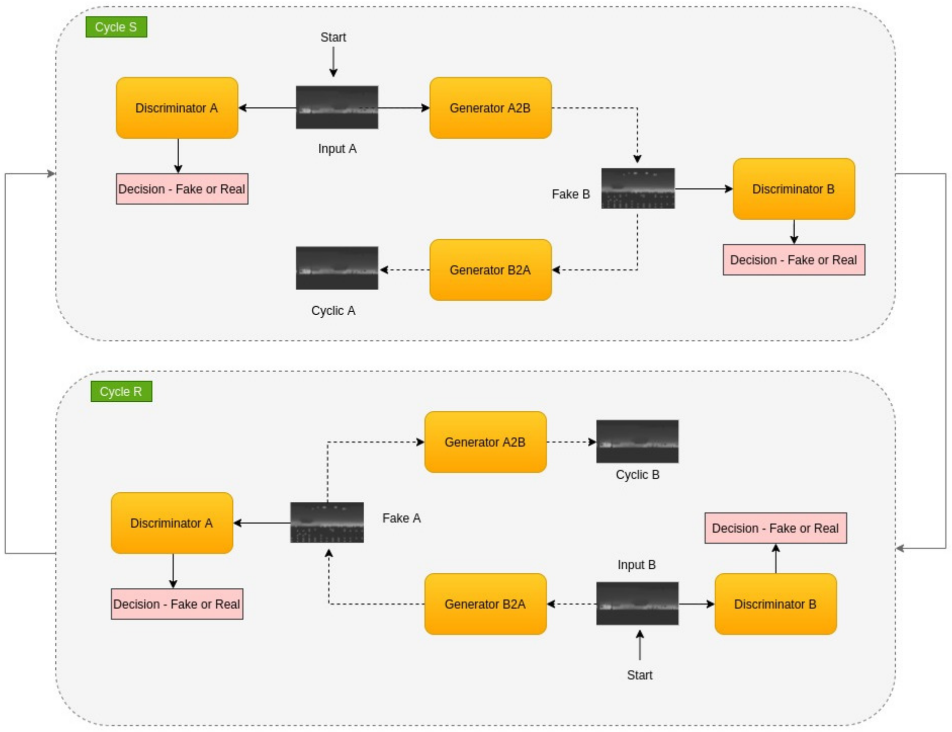


Figure 4.1.: CycleGAN overall workflow

- Output block: Finally, the upsampled data is processed to output the image of the same shape as the input

Discriminator Architecture: A bunch of convolutional layers along with instance normalization and leaky relu activations are used one after the other and at the end, a fully connected convolution layer followed by average pooling and flattening layers is used for classifying the images as fake or real. The discriminator architecture is depicted in the figure 4.3.

Formulation of the Losses:

A usual loss function for vanilla GAN generator consists of only the adversarial loss. Since CycleGAN is based on cyclic consistency, it introduces one more loss function called the cyclic consistency loss. The overall loss function of the network is the summation of adversarial loss and cyclic consistency loss. The normal generator is termed G while the reverse generator is F. The real data discriminator is termed as D_y while the recovered data discriminator is termed as D_x. A detailed description of training the CycleGAN can be seen in the figure 4.1

Adversarial Loss:

For normal Generator:

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

For reverse Generator:

$$\mathcal{L}_{GAN}\left(F, D_X, X, Y\right) = \mathbb{E}_{x \sim p_{data}(x)}\left[\log D_X(x)\right] + \mathbb{E}_{y \sim p_{data}(y)}\left[\log\left(1 - D_X(F(y))\right)\right]$$

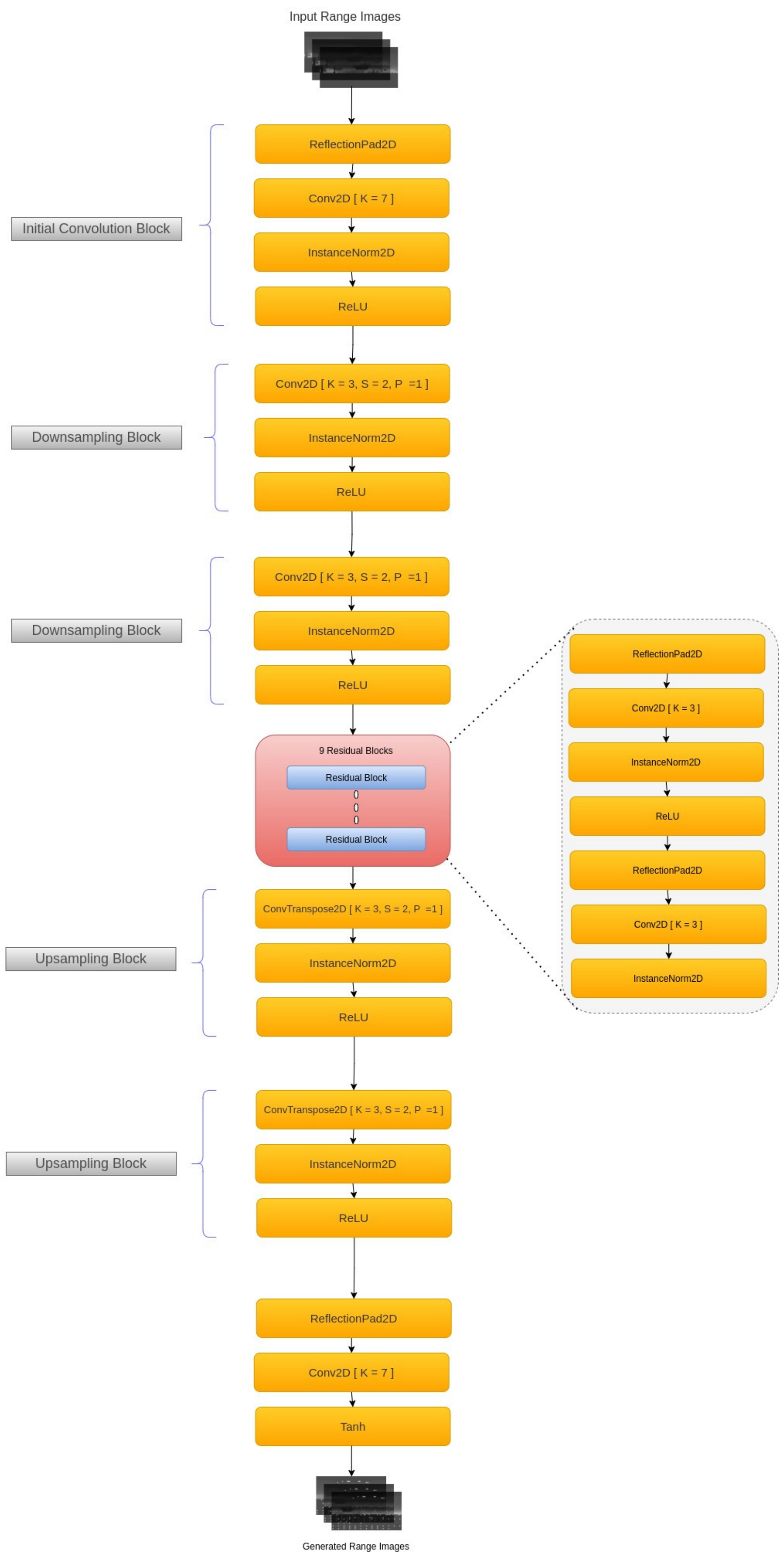


Figure 4.2.: CycleGAN Generator architecture

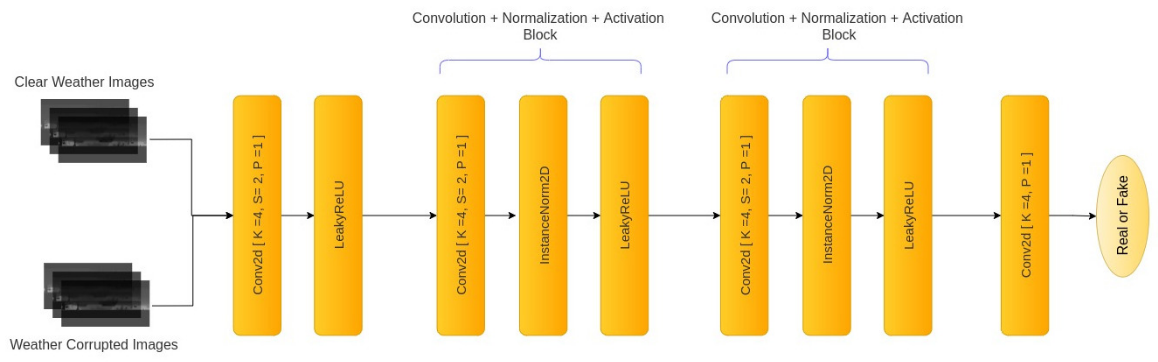

Clear Weather Images
Weather Corrupted Images
Convolution + Normalization + Activation Block
Convolution + Normalization + Activation Block
Conv2d [K =4, S= 2, P =1]
LeakyReLU
Conv2d [K =4, S= 2, P =1]
InstanceNorm2D
LeakyReLU
Conv2d [K =4, S= 2, P =1]
InstanceNorm2D
LeakyReLU
Conv2d [K =4, P =1]
Real or Fake

Figure 4.3.: CycleGAN Discriminator architecture

Cyclic Consistency loss: This is the quintessential part of the CycleGANs, two generators are trained simultaneously and the losses of each generator depends on the performance of the other generator and hence the losses are tied in a cyclic relationship.

$$\mathcal{L}_{R_Y}(G,F,Y) = \mathbb{E}_{y\sim p_{data}(y)}\left[R_Y\right] \quad = \mathbb{E}_{y\sim p_{\text{data}}(y)}\left[\|G(F(x)) - y\|_1\right]$$

where $R_Y = \|G(F(x)) - y\|_1$

$$\mathcal{L}_{R_X}(F,G,X) = \mathbb{E}_{x\sim p_{data}(x)}\left[R_X\right] \quad = \mathbb{E}_{x\sim p_{\text{data}}(x)}\left[\|F(G(y)) - x\|_1\right]$$

where $R_X = \|F(G(y)) - x\|_1$

The overall loss function will be the summation of all the losses mentioned above. The lambda is a multiplication factor that controls the extent of cyclic consistency is needed in the model

$$\begin{aligned}\mathcal{L}(G,F,D_X,D_Y) = \mathcal{L}_{GAN}(G,D_Y,X,Y) + \mathcal{L}_{GAN}(F,D_X,X,Y)\\ +\lambda\left[\mathcal{L}_{R_Y}(G,F,Y) + \mathcal{L}_{R_X}(F,G,X)\right]\end{aligned}$$

4.1.2. CNN Denoising

WeatherNet is one of the pioneers for deep learning-based LiDAR point cloud denoising. There is a plethora of research work demonstrating the capabilities of Convolution Neural Networks to learn the features of the image and also perform tasks such as object detection, semantic segmentation, and more. WeatherNet is inspired by the supervised semantic segmentation task using CNNs and performs the same on the LiDAR range images to classify each pixel on the range image to different categories of weather. The classification classes are as follows:

1. **Background** - This indicates the points on the range map that belong to the background and serves no purpose in representing the point clouds.
2. **Clear** - This indicates the points that belong to the clear weather and are prone to no weather-induced noise

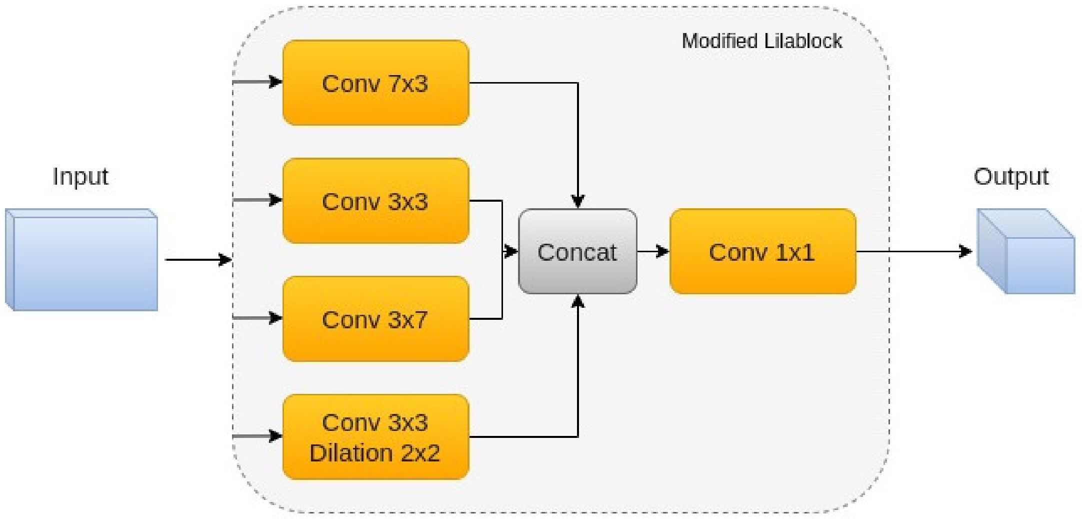


Figure 4.4.: The Modified LilaBlock architecture used in WeatherNet

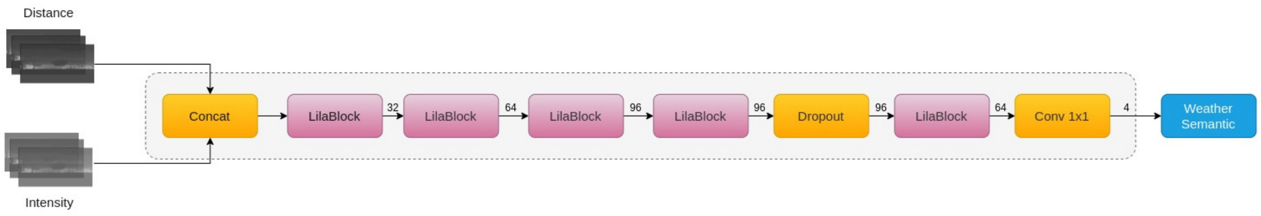


Figure 4.5.: The WeatherNet Architecture

3. **Fog** - This indicates the points that are scattered or affected by the fog in the environment
4. **Rain** - This indicates the points that are scattered and affected by the rain in the environment

In many of the research regarding LiDAR denoising, the distances of the points are used to infer the weather-affected outliers in the point clouds and the intensities of the points are ignored. However, the intensity information of the points might serve a bigger purpose in classifying them into different classes. Inspired by this concept of using intensity values for classifying points, WeatherNet uses both the distance and the intensity values of the points to classify the points into corresponding classes.

For denoising the LiDAR images, state-of-the-art Convolution Neural network-based semantic segmentation architectures are used for segmenting the sparse point clouds represented on a 2D plane. The WeatherNet is an efficient variant of the LiLaNet which is used for LiDAR segmentation

Lilanet This is considered to be a dedicated Convolution neural network architecture for a high-quality LiDAR-based semantic labeling. It makes use of a number of LiLaBlocks, which was inspired by GoogLeNet inception modules, stacked one after the other to downsample the input image and infer the labels. The LiLaBlock originally consists of convolution kernels of sizes 73, 37, and 33 in parallel. The output from all the kernels in the block is concatenated and the dimensions are reduced due to downsampling and hence the reduced dimensions of the feature space provide a compact representation of the input data. The convolution layers are followed by ReLu layers.

WeatherNet - WeatherNet uses the Lilanet with some modifications. As seen in the figure 4.4, LiLaBlock is modified to additionally contain a $2x2$ dilated convolution kernel of size $3x3$ in parallel with the rest of the kernels, this kernel is used to augment the receptive field by learning more information about the spacial vicinity. Furthermore, the depth of the network was reduced considering that we need 4 classes as outputs instead of 13 classes present in the LiLaNet model. Also, a drop layer is added to improve the generalization. The figure 4.5 shows the architecture for the semantic segmentation using WeatherNet.

Loss function The model prediction contains the classification details of the points on four different channels where the value of 1 indicates that the point belongs to that particular class and a value of 0 indicates it does not belong to that class. The semantic segmentation ground truth labels are on a single channel, and originally, the values such as 100 indicate clear points, 101 indicate rain points and 102 indicate fog points and by default 0 indicates the background. These values were remapped in such a way that 1 will indicate clear points, 2 will indicate rain-corrupted points and 3 will indicate fog-corrupted points. After the pre-processing, the ground truth labels and the predicted labels were compared using a cross-entropy loss that acts as the loss function for the network.

Training Details The model was trained on the Cerema dataset for 200 epochs with a batch size of 20, learning rate to $\alpha = 4.10^{-8}$ and learning rate decay of 0.90. Adam optimizer is used to optimize the overall network with the corresponding values being $\beta 1 = 0.9$, $\beta 2 = 0.999$ and $\epsilon = 10^{-8}$

4.1.3. Variational Autoencoder (VAE)

VAE, Variational Autoencoder, is a form of the traditional autoencoder. The main difference being that the VAE is regularized compared to the autoencoders. Similar to Autoencoder, it consists of 2 parts, an encoder network, and a decoder network. The encoder network maps the input data to a posterior distribution of latent codes of reduced dimensionality. The decoder network takes the latent codes as the input from the posterior distribution and aims to reconstruct the original input data. As shown in the figure 4.6, the working of a VAE is illustrated.

There are many research works that demonstrate the capability of VAE to perform image translation, reconstruction, and more. In the work, Deep Generative Modeling of LiDAR Data [9], the authors have employed VAE to work on LiDAR range image-based data. The VAE was used to demonstrate its capability to perform image reconstruction when some of the points on the input data were removed. Also, the model was evaluated for its denoising capabilities after the addition of additive Gaussian noise to the input.

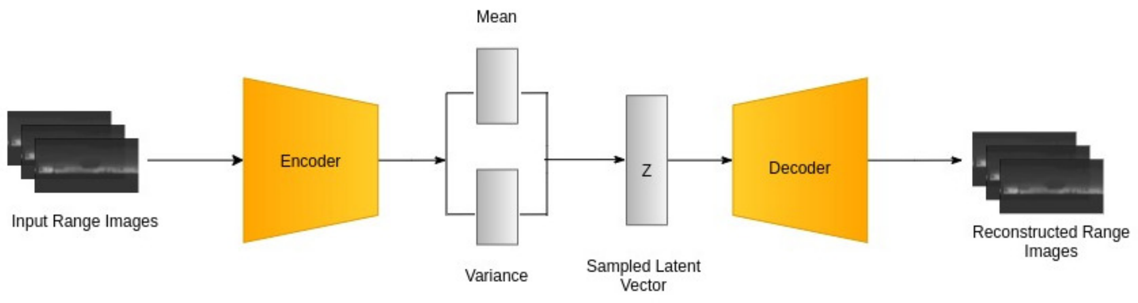


Figure 4.6.: A general variational autoencoder architecture

In this work, we employ a Variational Auto Encode and evaluate its performance to denoise the weather-induced noise such as scattering due to rain, fog, and snow on LiDAR range images. This model was evaluated for both the Seeing-through-fog and Cerema dataset.

Encoder Architecture This follows the architecture that was suggested in the work Deep Generative Modeling of LiDAR Data [9]. The encoder contains several convolution layers that downsample the inputs gradually and convert it to a lower-dimensional latent code. Batch normalization and activation layers with leaky ReLU activation are interleaved between the convolution layers. The architecture of the encoder can be seen in the figure 4.7

Decoder Architecture As suggested in the previous work, the decoder consists of 4 units of transpose convolution layer, batch normalization layer, and ReLu activation layer stacked one after the other. In the last unit, a transpose convolution layer followed by a tan activation layer is used to reconstruct the input data. The architecture of the decoder is depicted in the figure 4.8

Loss functions The loss, through which the model is trained, consists of 3 components.

The general objective of the variational autoencoder, KL_{Obj}, is seen as below

$$\mathcal{L}(\theta; x) = \mathbb{E}_{q(z|x)} \log p(x \mid z) - \mathrm{KL}(q(z \mid x) || p(z)) \leq \log p(x)$$

Reconstruction loss: This loss is nothing but the absolute difference between the original input and the reconstructed input. This can be termed as $Loss_{Reconstruction}$

Chamfer distance loss: This loss is the chamfer distance between the original input point cloud and the reconstructed input point cloud. This can be referenced as CD_{Loss}

The overall loss function of the network is as follows:

$$Loss = KL_{Obj} + \alpha Loss_{Reconstruction} + CD_{Loss}.$$

where α is a multiplier that controls the extent of involvement of the reconstruction loss in the overall loss function.

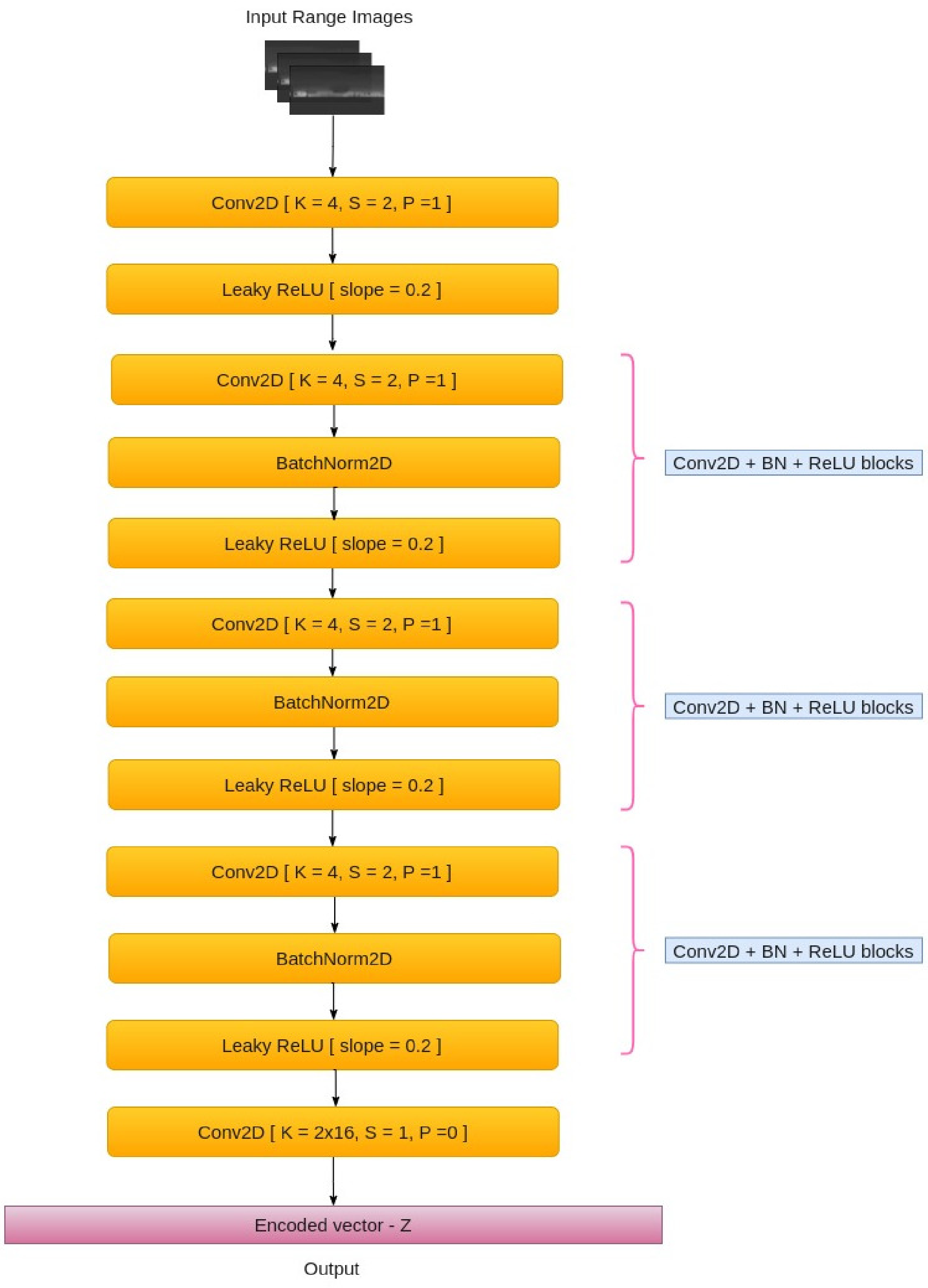

Input Range Images
Conv2D [K = 4, S = 2, P =1]
Leaky ReLU [slope = 0.2]
Conv2D [K = 4, S = 2, P =1]
BatchNorm2D
Leaky ReLU [slope = 0.2]
Conv2D + BN + ReLU blocks
Conv2D [K = 4, S = 2, P =1]
BatchNorm2D
Leaky ReLU [slope = 0.2]
Conv2D + BN + ReLU blocks
Conv2D [K = 4, S = 2, P =1]
BatchNorm2D
Leaky ReLU [slope = 0.2]
Conv2D + BN + ReLU blocks
Conv2D [K = 2x16, S = 1, P =0]
Encoded vector - Z
Output

Figure 4.7.: The encoder architecture of the VAE

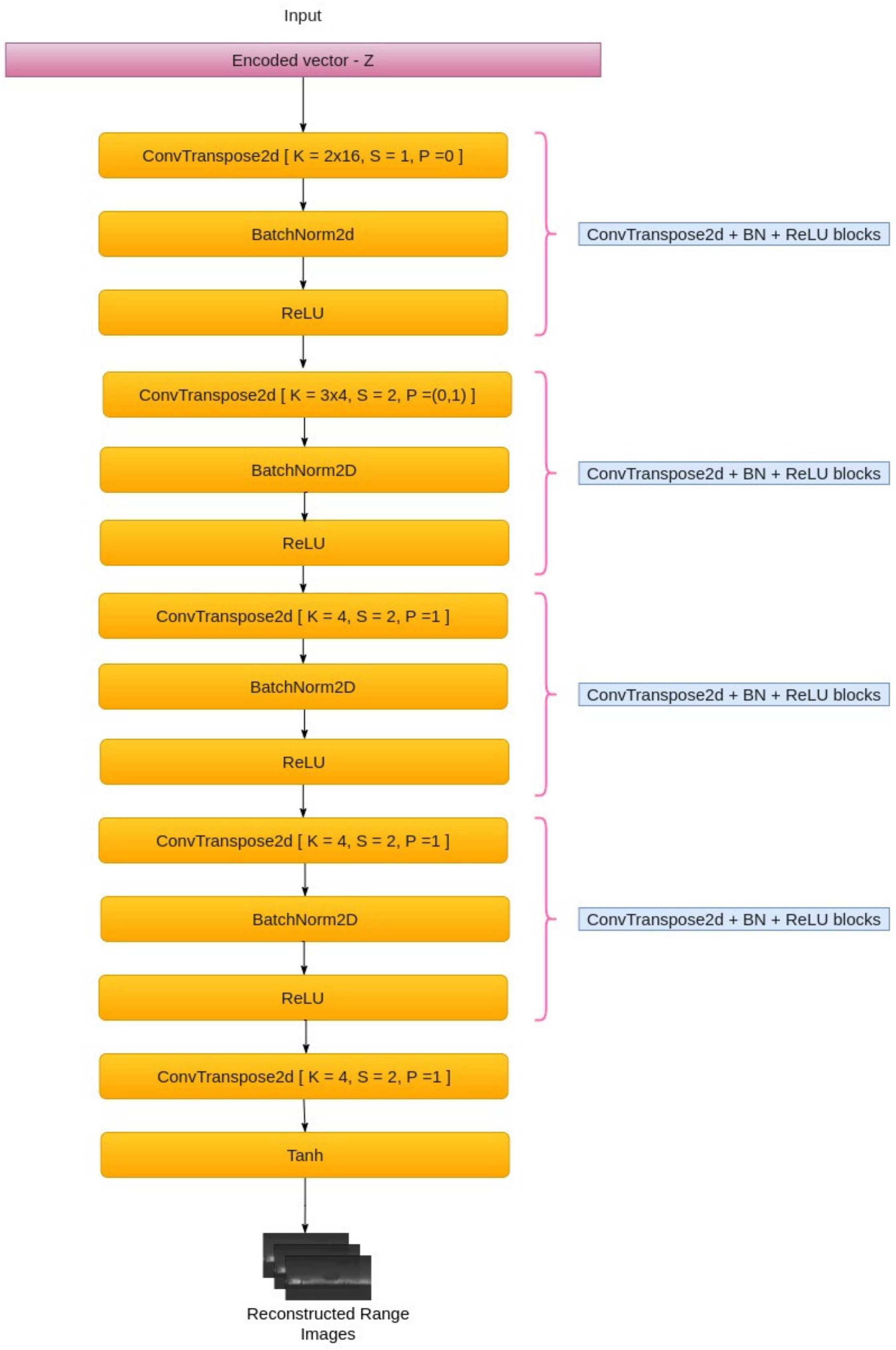

Input
Encoded vector - Z
ConvTranspose2d [K = 2x16, S = 1, P =0]
BatchNorm2d
ReLU
ConvTranspose2d + BN + ReLU blocks
ConvTranspose2d [K = 3x4, S = 2, P =(0,1)]
BatchNorm2D
ReLU
ConvTranspose2d + BN + ReLU blocks
ConvTranspose2d [K = 4, S = 2, P =1]
BatchNorm2D
ReLU
ConvTranspose2d + BN + ReLU blocks
ConvTranspose2d [K = 4, S = 2, P =1]
BatchNorm2D
ReLU
ConvTranspose2d + BN + ReLU blocks
ConvTranspose2d [K = 4, S = 2, P =1]
Tanh
Reconstructed Range Images

Figure 4.8.: The decoder architecture of the VAE

Training details The network was trained on the Seeing-through-fog and Cerema datasets separately. The size of the range image input and output considered is 32 x 400 and in spherical coordinate. The model was trained for 300 epochs with a batch size of 8 and later evaluated with respect to the chamfer's distance, earth mover's distance, and root mean square errors

4.1.4. Pix2Pix

In general, many of the problems encountered in image processing and computer vision would be to transform an image into another image. Previously, to perform an image translation between one image to another, one has to know the mapping function to translate the source image to the target image. Many works in image processing and computer vision focused on finding the hand-crafted methods to do the mapping between the two images. However, the crafted methods would be for specific tasks one has to find different methods for different tasks and this made the task of image translation difficult.

The advent of deep learning and, especially, generative adversarial networks paved the way for a better approach in image translation. The mapping function was no longer required to be meticulously crafted but can be learned using examples. Pix2Pix model [41] is one such front runner in image translation task that can take the paired source and target images and can learn the mapping function and avoid the need for devising handcrafted mapping functions

Pix2Pix model is a supervised image to image translation model based on the conditional Generative Adversarial Networks [89] and contains a generator and a discriminator. As the name suggests, the discriminator discriminates the real images from the fake images and the generator generates fake images to mislead the discriminator to classify the fake images as valid. The architecture of the generator and the discriminator can be seen below.

Generator: Pix2Pix model offers various types of generators such as Res-Nets and U-Nets with varying numbers of blocks. In this work, the Res-Net architecture with 9 blocks has to been used to generate fake images. The generator architecture can be seen in the figure 4.9

Discriminator: PatchGAN is used as a discriminator. Instead of penalizing by considering the entire image, patchGAN breaks down the image into patches of NxN and penalizes each patch of the image. This aids in better discrimination of real images from the fake images. The architecture of the PatchGAN can be seen in the figure 4.10.

Formulation of losses: The generator loss is can be divided into 2 - GAN loss and the reconstruction loss. GAN loss is the conditional adversarial loss and can be seen below.

$$\begin{aligned}\mathcal{L}_{cGAN}(G, D) =& \mathbb{E}_{x,y}[\log D(x, y)] + \\ & \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))]\end{aligned}$$

Reconstruction loss:

$$\mathcal{L}_{L1}(G) = \mathbb{E}_{x,y,z}\left[||y - G(x, z)||_1\right]$$

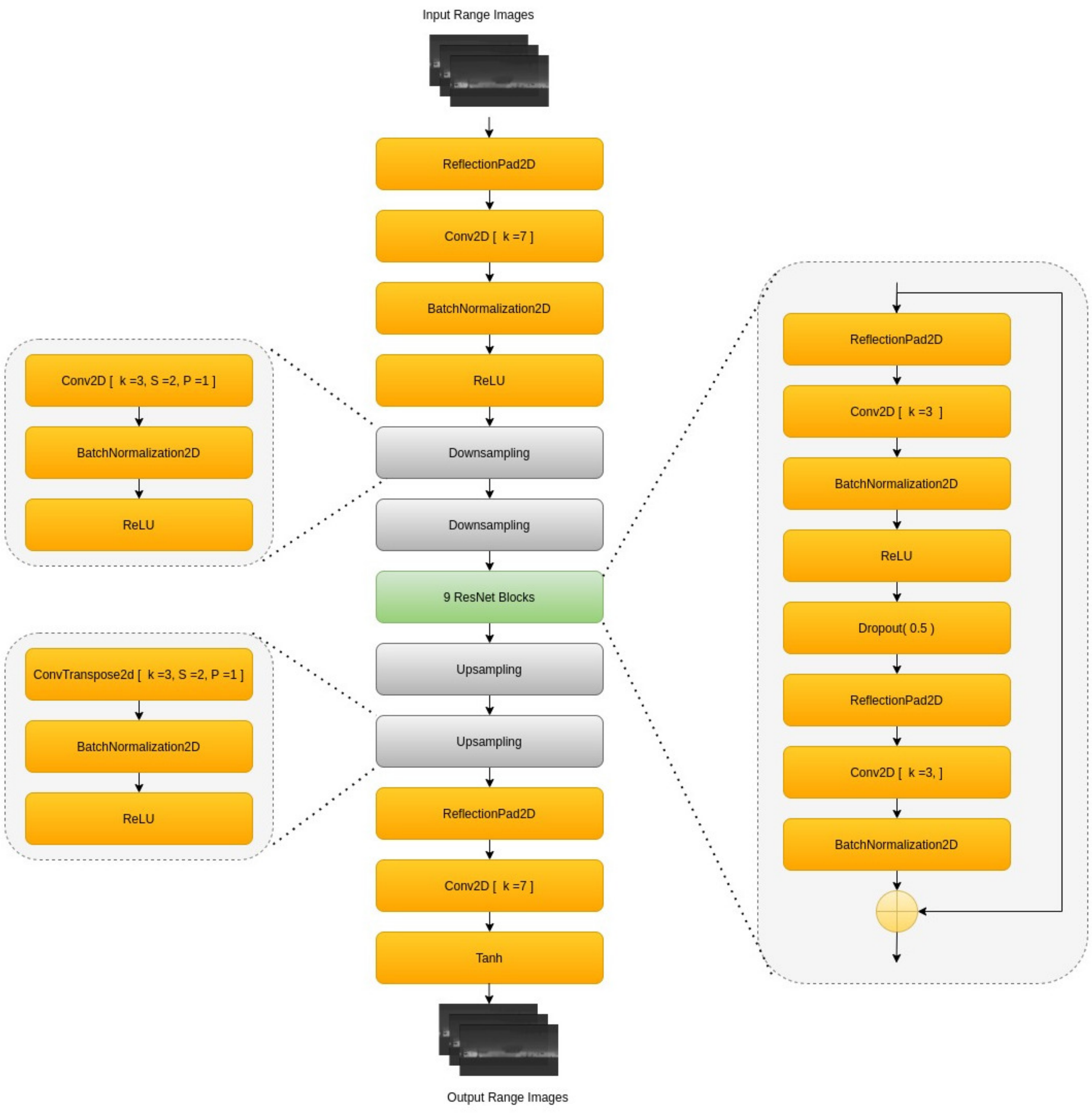


Figure 4.9.: The ResNet generator in Pix2Pix

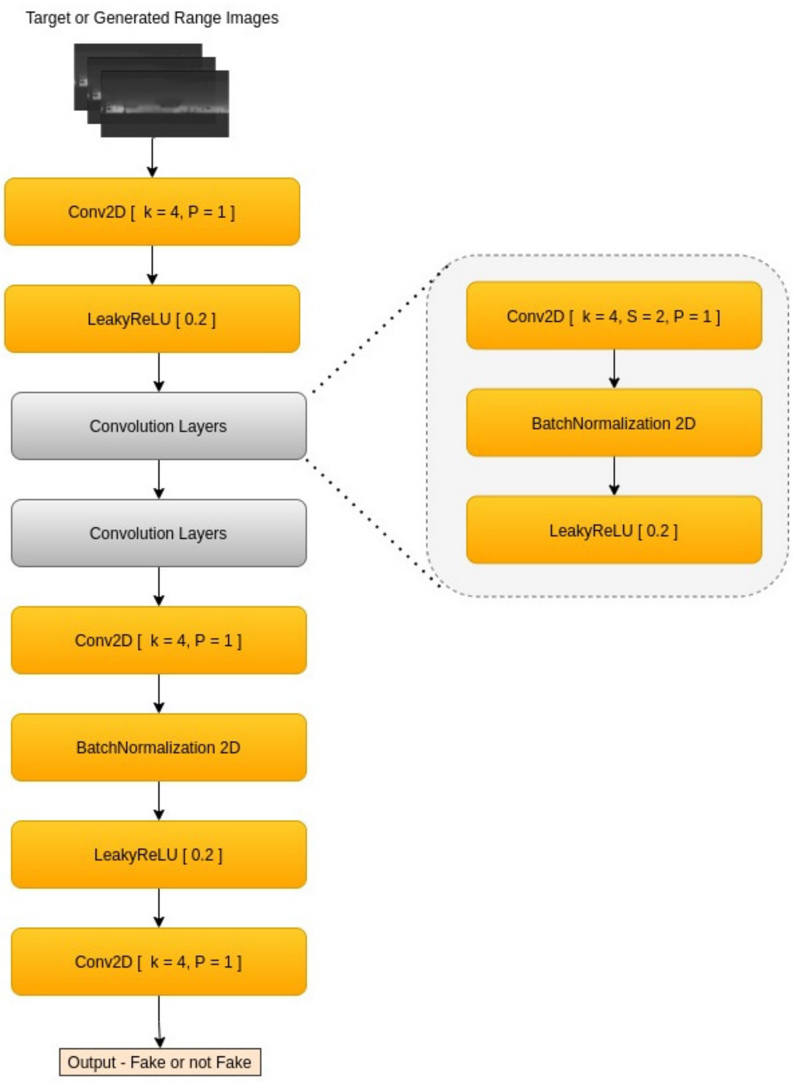


Figure 4.10.: The PatchGAN discriminator in Pix2Pix

Overall Generator loss:

$$\mathcal{L}_{Overall}(G) = \mathcal{L}_{cGAN} + \mathcal{L}_{L1}(G)$$

Training details The corrupted LiDAR images in the Cerema dataset were randomly paired with the LiDAR images that belong to the same scene but with no corruption. This pseudo-paired data was used to train the Pix2Pix model. The model was run for 400 epochs with a batch size of 8.

4.1.5. Dynamic Radius Removal Filter

There are some of the statistical methods for LiDAR point could denoising as mentioned in the previous section. Out of the many, dynamic radius outlier removal filter has been claimed to have a better performance on removing the noise in the LiDAR point cloud caused by snowfall, and hence we also implement a DROR filter as a statistical method to remove the noise from the seeing through fog dataset which not only contains snow corrupted lidar data but also the corruptions caused by rain and fog.

The implementation of the filter is an extension of the ROR filter, which works on the basis of the neighbors around a point in the point cloud. In Radius Outlier Removal (ROR) filter, a region of a fixed radius is searched around a point, and if there are neighbors less than a minimum number, then that particular point is considered to be an outlier and hence discarded. The major disadvantage of this filter is that in LiDAR scan, the farther the distance of the point from the origin, the sparser the neighborhood gets around that point although it's not an outlier. However, ROR filter considers this an outlier and discards the point. This is overcome in the DROR filter by using a radius that changes dynamically as the distance of the point from the origin changes and hence, in the farther regions of the lidar scan, the search radius increases to consider the sparser neighborhood.

The code snippet for the Dynamic radius outlier removal filter in python is shown below in 4.1

Listing 4.1: Conversion to spherical distance from the cartesian coordinates

```python
import numpy as np
from open3d import *
import math
def process(data):
    pcd_data = data[:,:3]
    source_data = geometry.PointCloud()
    source_data.points = utility.Vector3dVector(pcd_data)
    pcd_0 = source_data.voxel_down_sample(voxel_size=0.04)
    pcd_0_tree = geometry.KDTreeFlann(pcd_0)

    max_knn = 100
    radius_multiplier_ = 5
```

```
azimuth_angle_ = 0.04
min_neighbors_ = 3
min_search_radius_ = 0.3

final_points = []
total_points = len(pcd_0.points)
outlier_count = 0

for i in range(len(pcd_0.points)):
    dist_xy = math.sqrt( pcd_0.points[i][0]**2 + pcd_0.points[i][1]**2 )
    search_radius_dynamic = radius_multiplier_ * (azimuth_angle_ *
        math.pi / 180) * ( (dist_xy/5)**3 )

    if search_radius_dynamic < min_search_radius_:
        search_radius_dynamic = min_search_radius_
    [k, idx, _] = pcd_0_tree.search_hybrid_vector_3d(pcd_0.points[i],
        radius=search_radius_dynamic, max_nn=max_knn)

    if k < min_neighbors_:
        print("No enough neighbors {}, {}, {}!".format(i,
            pcd_0.points[i], k))
        outlier_count = outlier_count + 1
    else:
        final_points.append(pcd_0.points[i])

point_cloud = geometry.PointCloud()
point_cloud.points = utility.Vector3dVector(final_points)

visualization.draw_geometries([point_cloud])
```

This filter was implemented and qualitative evaluations were performed on the Seeing-Through-Fog dataset.

4.2. Proposed Approach

4.2.1. MaskGAN

As discussed previously, generative networks such as CycleGAN and VAE are used for unsupervised lidar range image translation to remove additive noise as well as to reconstruct some of the points that are lost in the lidar range map. In this way, these generative networks can be used to reconstruct an object in the lidar range image whose points are removed due to non-structural noise-induced to the lidar image such as the one during adverse weather where weather conditions such as rain, fog, or snow can cause random scattering of the lidar rays and hence can spoil the reading of the lidar sensor and in-turn corrupt the point clouds. However, the generative models mentioned above do not have any explicit idea about the noise-induced and hence would solely just do an image translation.

On the other, the WeatherNet model [7], used a convolution neural network-based model in a semantic segmentation task to classify the points in the lidar range image into various classes of weather such as clear, rain, and fog by taking a weather-induced noisy lidar data as the input. In this method, the points pertaining to weather such as fog or rain can be removed from the point clouds and only the clear points can be retained. However, the noisy points are just removed but clear points are not reconstructed and hence the output lidar image contains incomplete objects with missing points.

Some of the recent research work provides strategies to denoise the images in a self-supervised methodology. Noise2Noise [90] and Noise2Self [91] provide methods to remove the noise from the image while being trained on the same corrupted image. Similarly, Noise2Void [92] is one such method, where some of the pixels in an image are removed and the network is trained to generate these pixels in self-supervision. Our model takes the inspiration from the self-supervised denoising methods and with the problem that exists in the current state-of-the-art LiDAR denoising models and hence we propose an idea to overcome the problem.

The idea is to make the model aware of the weather-induced noise distributions while reconstructing the output lidar range image. In other words, we first detect the points that are prone to noise in the lidar data. We remove all the noisy points from the data, and we would be left with a lidar image that has a lot of missing points and incomplete objects. We would take this incomplete lidar data, and perform an image translation to the target distribution that contains clear lidar data and in the process, reconstruct the missing points on the filtered lidar data as per the clear weather distribution.

MaskGAN is based on a generative adversarial network model which is aware of noise. We use an auxiliary WeatherNet model to make MaskGAN aware of noise. The input noisy lidar data is first processed using the auxiliary weathernet model through which we generate a mask that removes the noise from the lidar data and retains only the clear points. The filtered data is passed onto a generative adversarial network. The GAN takes the filter input as the source distribution and maps it to the target distribution containing the lidar data that was not affected by weather and hence performs an unsupervised image translation from the incomplete filtered lidar input to the reconstructed clear weather output.

Generator architecture MaskGAN, unlike CycleGAN, uses one generator which takes the noisy input and outputs a denoised reconstructed output. The generator architecture is the same as the one used in CycleGAN. It uses the blocks as used by CycleGAN, i.e Initial Convolution block, Downsampling block, Residual block, Upsampling block, and Output block.

Discriminator architecture Similar to the generator, there is only one discriminator used in MaskGAN. The architecture of the discriminator is the same as the one mentioned in cycleGAN which contains a bunch of convolutional layers along with instance normalization and leaky ReLu activations are used one after the other and at the end, a fully connected convolution layer followed by average pooling and flattening layers is used for classifying the images as fake or real.

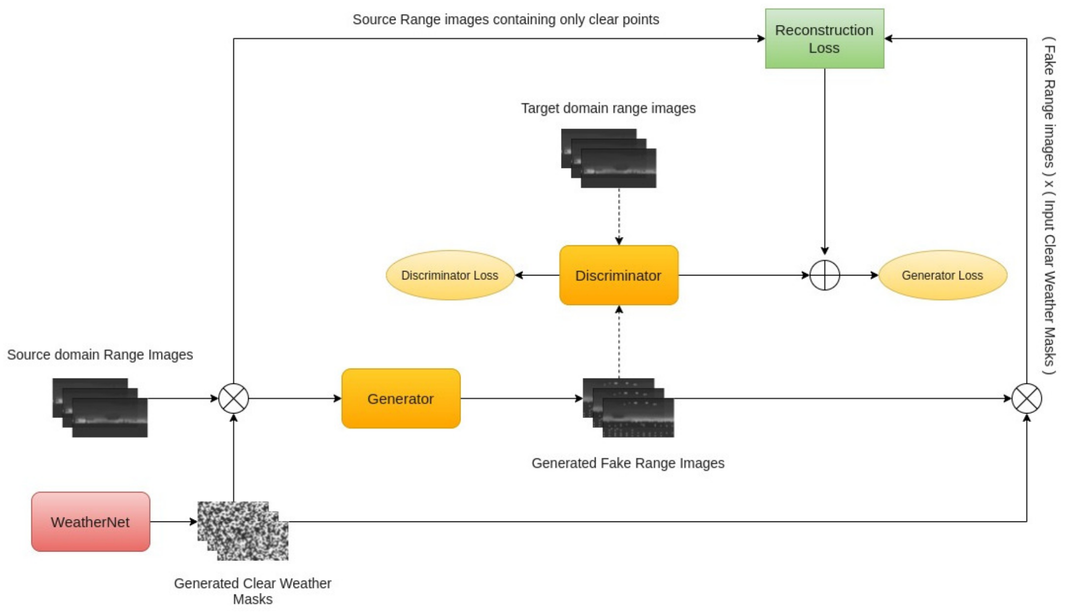

Source Range images containing only clear points
Reconstruction Loss
Target domain range images
Discriminator Loss
Discriminator
Generator Loss
(Fake Range images) x (Input Clear Weather Masks)
Source domain Range Images
Generator
Generated Fake Range Images
WeatherNet
Generated Clear Weather Masks

Figure 4.11.: MaskGAN architecture with simple reconstruction loss

Formulation of loss

Generator loss

There are two MaskGAN with different loss functions for the Generator and they can be seen as below

Simple Generator loss function This loss function can be divided into two components. The first one is the regular adversarial loss shown below.

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

where D_y is the discriminator, and G and x is the input image distribution

The second is the reconstruction loss to which the mask is applied. The input image is multiplied by the mask to get the points that belong only to the clear weather. The fake generated image is also multiplied by the same mask to get the points that refer to clear weather in the input image. The reconstruction is considered for the points that are made available after masking the input and the fake generated images as seen below:

$$\mathcal{L}_{Recons}(G, X) = \mathbb{E}_{x \sim p_{data}(x)}\left[\|xM(x) - G(x)M(x)\|_1\right]$$

where $M(x)$ is the function that generated the clear weather mask from the input image x

The overall loss of the generator is L_{gen}, where

$$L_{gen} = L_{GAN}\left(G, D_Y, X, Y\right) + L_{Recons}(G, X)$$

Compound Generator loss function This is a slightly advanced loss function that combines multiple losses to train the generator. The losses are mentioned below:

Total Variation Loss is a regularization loss [93] that promotes piece-wise smoothness in the output image. This sums up the absolute differences for the nearby pixel values in the input and, in turn, measures the amount of noise present in the input. The product of the fake generate image and the clear weather mask is given as the input to the TV loss function. This can be referred to as TV_{LOSS}

Adversarial loss is the binary cross-entropy loss between the discriminator produced labels and the ideal image label, which is 1. The equation is the same as the L_{GAN}

Image reconstruction loss is the mean square error loss between the product of the input image and the clear mask and the product of the fake generated image and the clear mask.

$$\mathcal{L}_{MSE}(G, X) = \mathbb{E}_{x \sim p_{data}(x)}\left[xM(x) - G(x)M(x)\right]^2$$

The overall loss function is the summation of all the losses

$$L_{gen} = L_{MSE}(G, X) + \alpha L_{GAN}(G, D_Y, X, Y) + \beta TV_{Loss}$$

where α and β are the multipliers with values 0.001 and 2e-8 respectively

4.2.2. U-Net based Pix2Pix

This model acts as an improvement to the general Pix2Pix mentioned earlier, wherein the U-Net model is taken to be the generator of the supervised GAN. The general architecture remains the same as the Pix2Pix architecture. However, the model uses an enhanced loss function that the one introduced in Pix2Pix.

Generator The generator uses is the U-Net which has a double convolution layer followed by four layers of downsampling resulting in a flattened vector. The four upsampling models not only take the input from the previous layer but also from its downsampling counterpart and generated the output image. This architecture is depicted in the figure 4.12

Discriminator This model uses the same PatchGAN discriminator used by Pix2Pix as shown in figure 4.10

Formulation of losses The loss used in this model is the same as the compound loss mentioned in the MaskGAN except for the masking

$$\mathcal{L}_{Mse}(G) = \mathbb{E}_{x,y}\left[y - G(x)\right]^2$$

where y is the ground truth image and x is the input image, G and D_y are generator and discriminator respectively

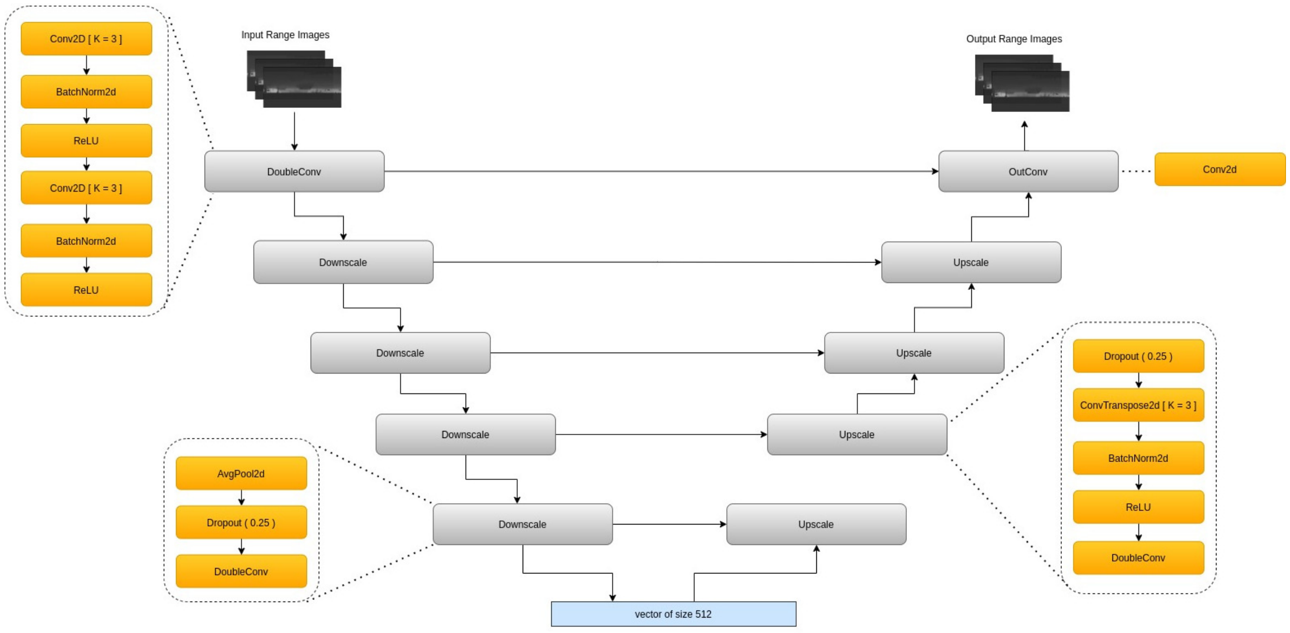


Figure 4.12.: The U-Net architecture

$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_{x,y}[\log D(x, y)] + \\ \mathbb{E}_{x,z}[\log(1 - D(x, G(x, z))]$$

and TV_{Loss} on the fake image generated

The overall loss function is given by

$$L_{gen} = L_{MSE}(G, X) + \alpha L_{GAN}(G, D_Y, X, Y) + \beta TV_{Loss}$$

where α and β are the multipliers with values 0.001 and 2e-8 respectively

Training details The model is trained on the pseudo paired images of the Cerema dataset for 400 epochs and a batch size of 8.

5. Evaluations

5.1. Metrics

Earth Mover's Distance (EMD) Earth Mover Distance [94] is a measure used to describe the similarity between two multidimensional distributions. Originally, it solves a transportation problem that attempts to transform one set into the other. It is one of the few metrics that are permutation invariant and can be applied to unordered point sets and it has been used as a metric to measure the similarity between two point clouds in many works [95] [96] [67].

Given two 3 dimensional point clouds P and Q representing the LiDAR data. The earth mover's distance between P and Q can be calculated if P and Q are of equal size, i.e $|P| = |Q|$. Then, the equation to calculate EMD is as follows:

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \rightarrow S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2$$

where P and Q represent the point clouds and $\varphi : P \rightarrow Q$ represents a bijective mapping from the source point cloud to the target point cloud.

The EMD distance addresses the assignment problem, which is an optimization problem. The ideal bijection is unique and invariant under the point's infinitesimal movement except for a zero-measure subset of point set pairs. As a result, EMD is completely differentiable.

In practice, even on a GPU, precise EMD computation is too expensive using deep neural networks and hence to be used frugally. Recent work is being done with entropic EMD or Wasserstein distances to accelerate EMD calculation. But even the entropic approach for approximating the EMD is still expensive than Chamfer's distance and hence CD is still used widely. Therefore, there is still a need to develop point cloud metrics that have the fine-grained properties of EMD but has the computational complexity of Chamfer's Distance.

In our supervised models, EMD is one of the main metrics we used to compare the point clouds by finding the average of the EMD distances of all the point clouds

Chamfer's Discrepancy (CD) Many autoencoders are developed in recent years [97] that are trained mainly on Chamfer's discrepancy (CD) or Earth mover's distance (EMD) as these distance metrics are the popular choices for unordered point set similarity measurements. Although it is claimed that EMD is better for 3D point cloud reconstructions task compared to CD [97], [98], Chamfer's discrepancy is more preferred [99] as it is computationally light and can use used in the loss functions of a deep neural network with less overhead and hence used widely in deep learning on tasks related to point clouds. Chamfer's discrepancy is sometimes also regarded as Chamfer's distance, however, CD is a pseudo-distance metric, not a distance metric [98].

Consider two point clouds P and Q, the Chamfer's distance can be written as follows:

$$d_{\mathrm{CD}}(P,Q) = \frac{1}{|P|}\sum_{x\in P}\min_{y\in Q}\|x-y\|_2^2 + \frac{1}{|Q|}\sum_{y\in Q}\min_{x\in P}\|x-y\|_2^2$$

Chamfer's discrepancy only concerns a point's nearest neighbor, not the distribution of those nearest points, this is depicted in the above formula as a min function. As a result, as long as x and y are close, Chamfer's disparity between them is modest, even if their associated distributions are different. Also, one can use the spatial nearest neighbor search algorithms using the KD tree which can make the calculation of CD faster and since the range search is not dependent on the points, the search algorithm can be parallelized. Furthermore, Chamfer's distance doesn't require both the point clouds P and Q to have the same number of points and hence advantageous when it comes to comparing two different distributions.

However, it was also shown that Chamfer's discrepancy doesn't perform well sometimes [97] as it cannot distinguish between a true sample and a bad one because of its inherent non-discriminative nature.

Also, there are variants of Chamfer's discrepancies that are used in some of the works. In this, the summation term from the previous equation is replaced by the max function as shown below.

$$\begin{aligned} d_{\mathrm{MCD}}(P,Q) = \max\{&\frac{1}{|P|}\sum_{x\in P}\min_{y\in Q}\|x-y\|_2^2 \\ &\frac{1}{|Q|}\sum_{y\in Q}\min_{x\in P}\|x-y\|_2^2\Bigg\} \end{aligned}$$

However, we use the non-modified version of CD in the measurements of supervised image translation models mentioned in the rest of the work.

Root Mean Square Error (RMSE) The root-mean-square error (RMSE) is a commonly used metric for comparing the disparities between a model's predicted and observed values. It represents the square root of the average value of all the squared differences between the observed value and the predicted value.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2$$

$$\mathrm{RMSE}(\hat{\theta}) = \sqrt{\mathrm{MSE}}$$

This is a metric that is used in calculating the similarity of two paired LiDAR range images. The higher the value, the lower is the similarity. This is also widely used as a loss function for the discriminator in GANs and also used as a loss metric for reconstruction losses in some of the GANs architecture [12]

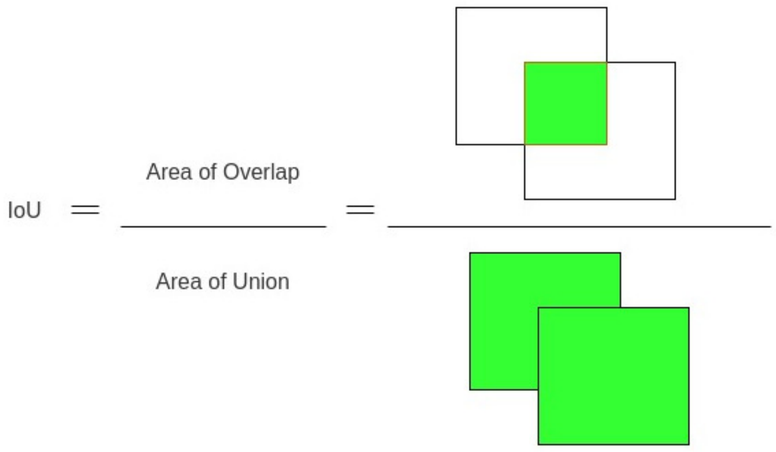


Figure 5.1.: Intersection over Union

Mean Intersection Over Union (mIoU) Segmentation is the process of dividing an image into segments. In instance segmentation, an image is divided into segments where the required objects are differentiated using the bounding boxes around the objects. In semantic segmentation, the image is segmented into classes where the generated mask indicates to which category the corresponding pixels belong. Both these tasks - Instance segmentation and semantic segmentation - require an evaluation metric that could tell if the model is training or performing well. Following this need, one way to evaluate is to use the overlap of the predicted bounding boxes or the classes with the ground truth bounding boxes or the classes. And, this initiated a new metric called the Intersection over Union

One of the most often used metrics in semantic segmentation as well as in instance segmentation is the Intersection over Union (IoU), also known as the Jaccard Index. The IoU is a very simple, highly useful metric.

Intersection over Union is the region of overlap between the ground truth segmentation and the predicted segmentation divided by the area of union between the ground truth and the predicted segmentation. The mean of the summation of the intersection over union values of all the samples is called the mean Intersection over Union. This can be seen in the figure 5.1

The value of IoU or mIoU ranges from 0 to 1. The higher the value of mIoU, the better is the performance of the model. As seen in the figure 5.2, the IoU for A is given to be 0 because the two images do not overlap. This indicates the lowest value of the metric. In example B, we can see that it is certain about overlap and the IoU is 1/7. Similarly, in C, all the regions of the 2 images overlap, and hence the IoU is 1, which is the highest.

In our model, essentially in the WeatherNet model, we use the mean Intersection over Union metric to use as the metric for loss function when training the CNN-based semantic segmentation model as well as an evaluating metric to measure the denoising performance of the segmentation model.

Frechlet Inception Distance (FID) In comparison to other networks, evaluating Generative Adversarial Networks (GANs) is extremely difficult due to its instability in training and insufficient accuracy of the existing methods. Furthermore, evaluating the quality of GANs

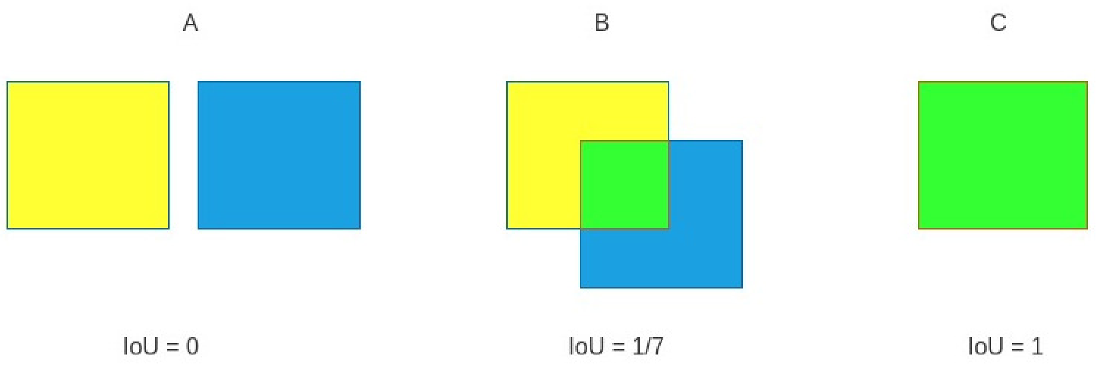


Figure 5.2.: Illustration of IoU calculation

is critical since it can assist us in selecting the appropriate model, determining when to end training, and determining how to enhance the model. Frechlet Inception Distance (FID) is one of the performance metrics used to assess the quality of GANs.

The Frechet Inception Distance, or FID, is a metric for assessing the quality of generated images that were designed expressly to assess the performance of generative adversarial networks. Martin Heusel et al. devised and used the FID score in their paper [100] "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium" published in 2017.

FID score is an improvement over the existing Inception Score. The inception score(IS) measures how well Inception v3 [101] recognizes a series of generated images as one of 1,000 recognized items. However, the Inception score(IS) doesn't take the statistics of the target images to compare with the generated images, and hence FID was introduced to overcome this drawback

The distance between the feature vectors of real images and the feature vectors of fake images is calculated using the FID metric. The lower the FID score, the higher the quality and similarity of the images generated by the generator are to real ones.

To calculate the FID, at first, the pre-trained Inception v3 model is used and the last layer of the model is removed. The output is considered to be the activation vector from the last global spatial pooling layer of the Inception v3 model. The output of this layer contains 2048 activation for each image and hence this is considered to be a feature vector of size 2048.

Feature vectors are calculated for both the target image and the fake generated image. The FID score is then calculated using the below-mentioned formula

$$\mathrm{d}^2 = ||\mathrm{mu}_1 - \mathrm{mu}_2||^2 + \mathrm{Tr}\left(\mathrm{C}_1 + \mathrm{C}_2 - 2\sqrt{(\mathrm{C}_1\mathrm{C}_2)}\right)$$

where the square of d is the FID score. mu_1 and mu_3 are the means of the feature vector of the target image and the fake images respectively. C_1 and C_2 are the co-variance matrix of the real image and the fake image respectively. Tr stands for the trace linear algebra operation, which is the sum of the elements along the square matrix's principal diagonal.

The operation of the FID calculation is depicted in the figure 5.3

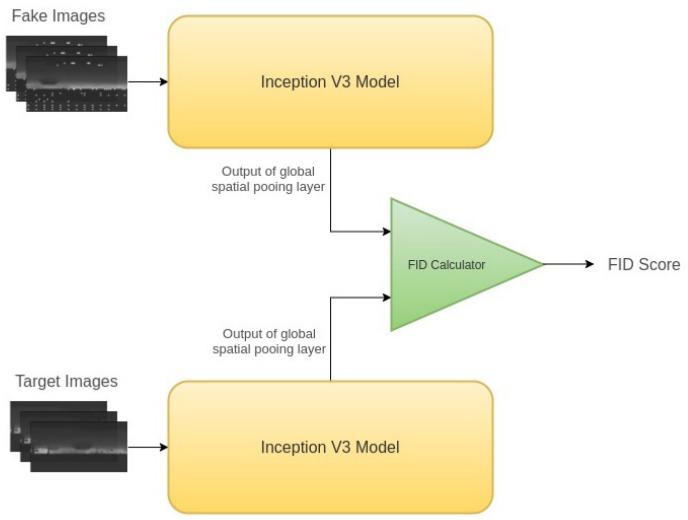


Figure 5.3.: FID score calculation

The FID metric can not only be used between two specific images but also between a range of images or image distributions. In the models we developed in this work, we use the Frechlet Inception Distance to evaluate the distance between the generated fake clear weather range image distributions and the real clear weather target range images.

Percentage of Points (POP) A lot of metrics mentioned above have been widely used for the supervised image to image translation task and are found to be very effective. The main advantage with respect to the supervised learning stems from the availability of the ground truth labels for the comparison with the predictions. However, when considering the realm of unsupervised learning, these metrics tend to be not as effective as for the supervised learning. Additionally, in the task of image denoising, there are no suitable metrics that can evaluate the presence or the absence of the noise accurately. Hence, there is a need for newer metrics that can evaluate the unsupervised models, pertaining to image denoising tasks, with a higher confidence level. In the same aspect, we introduce a novel metric called the Percentage of Points (POP) to evaluate the unsupervised model performance with respect to, not only the LiDAR image denoising but also with respect to the reconstruction capabilities of the model.

Percentage of Points (POP) is a metric that can give the percentage of the image that is prone to certain conditions such as noise and non-noise. In our work, the noise is induced by the weather and can be further divided into fog and rain. Also, the metric can further differentiate between the background points and the clear points. And hence the percentage of points (POP) can be calculated for the background points, clear points, points corrupted by fog, points corrupted by rain.

To enable the usage of POP, we use an auxiliary WeatherNet [7] unit which is trained on Cerema dataset and contains weather conditions such as clear, fog, and rain. Initially, the pre-trained WeatherNet model is used on the input LiDAR range image of the same size as Cerema dataset, $32x400$, to figure out the POP in various categories mentioned - background, clear, fog, and rain. This input image is further passed into the generator, where a fake denoised image is generated. The fake generated image is further evaluated by passing it as a input to the WeatherNet model, where the POP for all the categories in the Fake image are calculated and compared with the POP calculated for the source input.

The POP is calculated as follows:

1. $POP_{Background}$: The number of points that are equal to the value of 1 in the 1st channel of the mask, predicted by the auxiliary WeatherNet model for a given input, is divided by the overall size of the input, which is $32x400$ in the case of Cerema dataset.
2. POP_{Clear}: The number of points that are equal to the value of 1 in the 2nd channel of the mask, predicted by the auxiliary WeatherNet model for a given input, is divided by the overall size of the input, which is $32x400$ in the case of Cerema dataset.
3. POP_{Rain}:The number of points that are equal to the value of 1 in the 3rd channel of the mask, predicted by the auxiliary WeatherNet model for a given input, is divided by the overall size of the input, which is $32x400$ in the case of Cerema dataset.
4. POP_{Fog}: The number of points that are equal to the value of 1 in the 4th channel of the mask, predicted by the auxiliary WeatherNet model for a given input, is divided by the overall size of the input, which is $32x400$ iin the case of Cerema dataset.

As an evaluation of the performance of the model, the ratio of the mean POP (RPOP) of all the fake generated images and the source images can be calculated. Ideally for a good model that performs denoising as well as a reconstruction of point clouds, the lesser the RPOP of rain, fog and background and higher the RPOP of clear points, the better the performance of the model, i.e. $RPOP_{Bg} < 1, RPOP_{Rain} < 1, RPOP_{Fog} < 1$ and $RPOP_{Clear} > 1$ for a ideal model

A general formula of the ratio of percentage of points is shown below, this can be individually considered for each weather category.

$$RPOP = \frac{POP_{fake}}{POP_{Input}}$$

5.2. Evaluation Methodology

This section describes the various models that are trained and evaluated with respective various configurations of losses and datasets. In the beginning, we see the training and evaluations methods of unsupervised base models and the proposed unsupervised model. And later we will see the evaluation methods of the supervised models.

CycleGAN is one of the unsupervised base models we use for LiDAR denoising. There are two configurations of this model. In one configuration, we train it on the Seeing-Through-Fog dataset, and in the other configuration, we train it on the Cerema dataset. The model was trained using the adversarial loss, reconstruction loss, and cyclic consistency loss. To evaluate the model, we use the Fréchet inception distance (FID) between the generates fake images distribution and the clear weather target images distributions. Additionally, a pre-trained auxiliary WeatherNet model is used to find out the percentage of points in the input and the generated image that belongs to background data, clear weather, corrupted by fog, and corrupted by rain. Using this, we can also evaluate the CycleGAN model using the Ratio of Percentage of Points (RPOP) metric.

Variational Autoencoder is also one of the unsupervised base methods that are used for LiDAR denoising. VAE is trained and evaluated on the Seeing-through-fog dataset. VAE is trained on a loss function that contains the components of the KL divergence, reconstruction loss, as well as Chamfer's divergence. The model was evaluated by finding the FID score that was calculated between the generated output and the real output range image distributions.

The WeatherNet model is based on the semantic segmentation task that is trained only on the Cerema dataset. This uses ground truth labels to train the model and hence serves as a supervised base model. This was trained on the mean Intersection over union metric using a cross-entropy loss. The model was evaluated with the mean Intersection over union metric on the range images as well as the earth mover's distance and chamfer's distance between the input point clouds and the denoised point clouds. The FID score was calculated between the input image distributions and the denoised output image distributions.

The MaskGAN is trained and evaluated on the two datasets - the Cerema dataset and the Seeing-Through-Fog dataset. The model is evaluated with the FID score as well as the ratio of Percentage of points (RPOP). The MaskGAN configurations can be divided into two variants based on the loss functions used and both variants were evaluated.

Furthermore, a supervised image translation was made possible by the pseudo paired Cerema dataset. Pix2Pix on ResNet was trained and evaluated with respect to EMD and FID metrics. Additionally, a UNet based generator with custom loss function was also trained and evaluated.

5.3. Results

At first, we perform a qualitative visual analysis of all the models mentioned, and in the second half of the section we shall discuss the quantitative analysis that consists of the values of various metrics mentioned in the previous section

5.3.1. Qualitative Analysis

To understand the quantitative analysis, we make use of two of the visualization of LiDAR data. The first one is the Bird eye view of the LiDAR scan, the second one would be the range images of the LiDAR scan in spherical coordinates.

Dynamic radius outlier removal (DROR) This is a statistical filter that was implemented and used as a qualitative reference for the filtering quality. The filter was applied on samples of the Seeing-Through-Fog dataset and the noise removal capability of the filter can be seen in the figure 5.4

CycleGAN This model output's the range images of the input, sample clear weather, fake generated output as well as the input reconstruction from the fake image. The range images are again plotted back to 3D point clouds and the performance of the model was verified. The model was evaluated for both Seeing-Through-Fog and Cerema datasets.

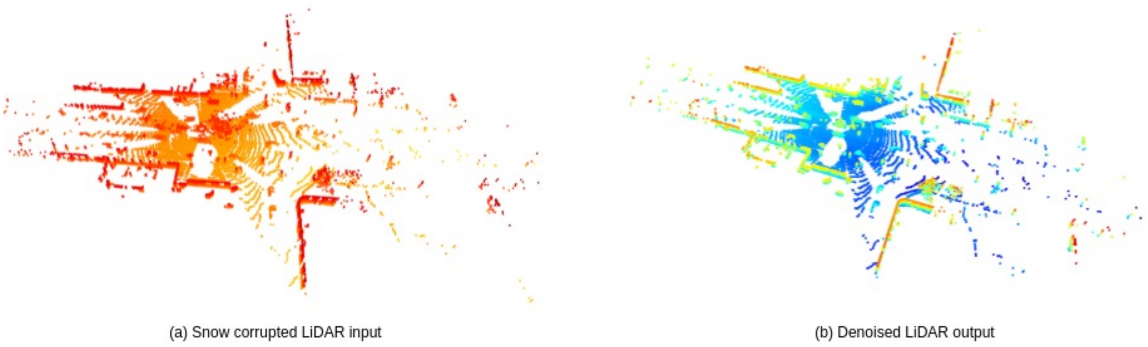
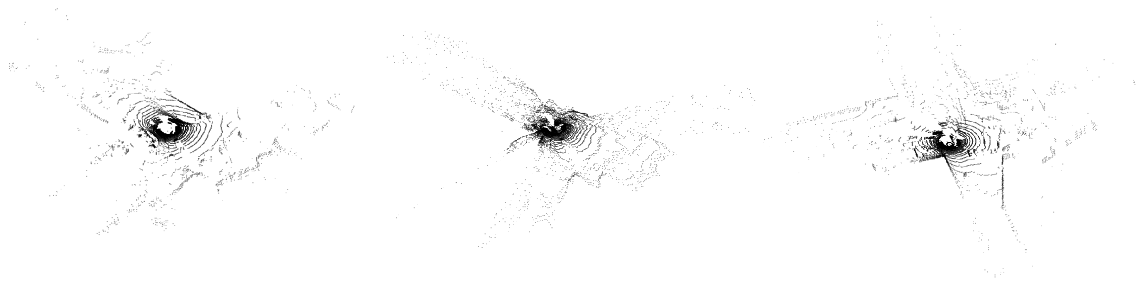


Figure 5.4.: Dynamic Radius Outlier Removal filter on Seeing-Through-Fog sample LiDAR scan

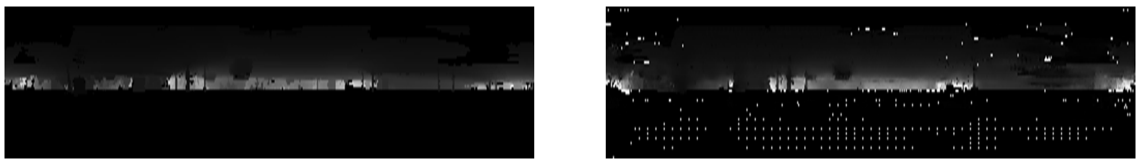


(a) Input (b) Fake (c) Sample Target

Figure 5.5.: CycleGAN - Point Clouds of a LiDAR scan in Seeing through fog dataset

A sample 3D point clouds of the Seeing-Through-Fog dataset can be seen in figure 5.5. It can be seen that the model tries to generate fake point cloud similar to the target clear weather point cloud. The generated a fake point cloud has the same shape and objects present in the input point cloud. However, as seen in sub-figure (b), the generated points sometimes do not entirely adhere to the scan line pattern present in the real LiDAR point clouds. The model performance can be better visualized with the help of the corresponding range maps. As seen in the figure 5.6 that shows the range maps for corrupted input and the fake generated image. The generated fake range map contains the objects present in the input. However, the range map contains some artifacts and this is because the image lacks a certain amount of scan line uniformity that's present in the regular LiDAR scans.

A sample 3D point clouds of the Cerema dataset can be seen in figure 5.7. It can be seen that the model tries to generate a fake point cloud similar to the target clear weather point cloud, however, the scan lines are closer and denser than the sample target image. Better analysis



(a) Input (b) Fake

Figure 5.6.: CycleGAN - Range Map of a LiDAR scan in Seeing through fog dataset

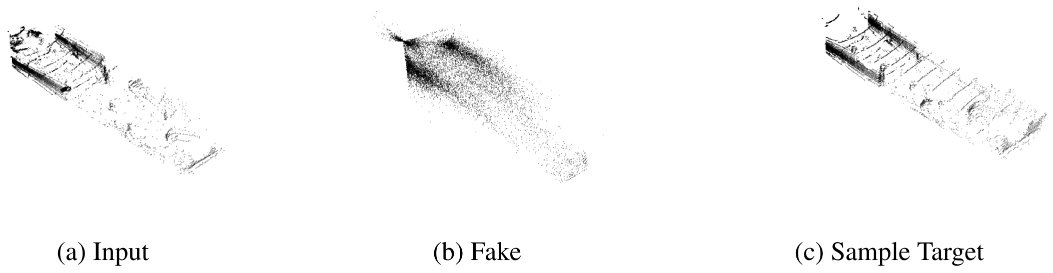

(a) Input (b) Fake (c) Sample Target

Figure 5.7.: CycleGAN - Point Clouds of a LiDAR scan in Cerema dataset

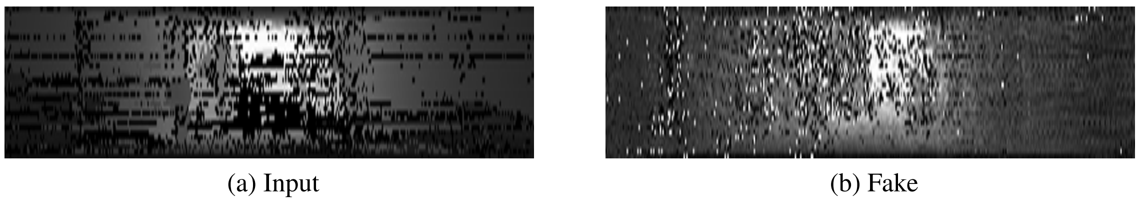

(a) Input (b) Fake

Figure 5.8.: CycleGAN - Range Map of a LiDAR scan in Cerema dataset

can be made by looking at the corresponding range maps in fig 5.8. The noise present in the input range image has been reduced in the fake generated range image. However, similar to the Seeing-Through-Fog dataset, there are checkerboard artifacts present in the fake range image.

Variational Autoencoder (VAE) The input range images, the reconstructed input, and the sample clear weather images are reconstructed into point clouds and visualized in the figure 5.9. The model was evaluated on the Seeing-Through-Fog dataset. As seen, the model performance on LiDAR point reconstruction is not effective. This can also be seen by visualizing the range images as shown in the figure 5.10 where the input contents are lost in the generated fake range image.

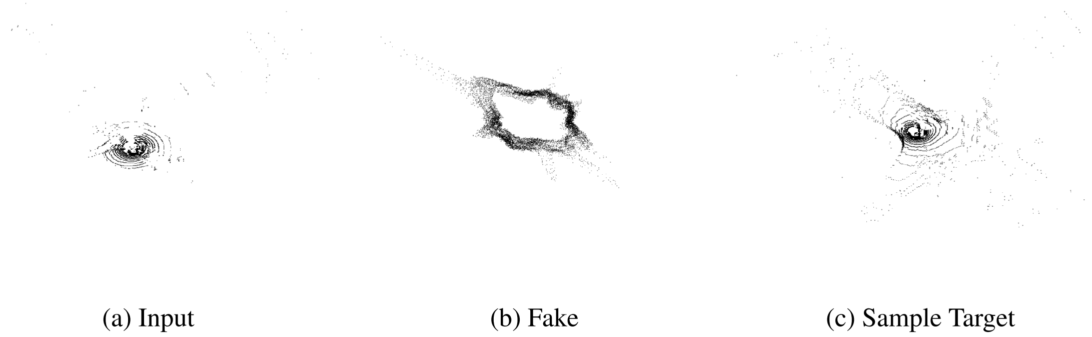
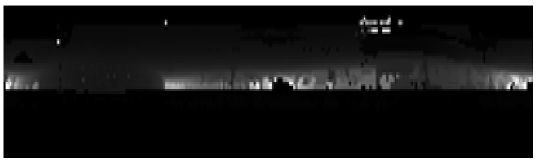

(a) Input (b) Fake (c) Sample Target

Figure 5.9.: VAE - Point Clouds of a LiDAR scan in Seeing through fog dataset


(a) Input

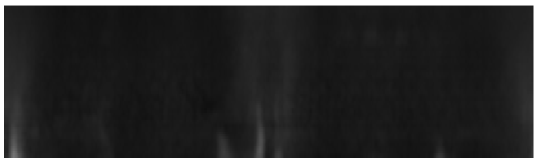
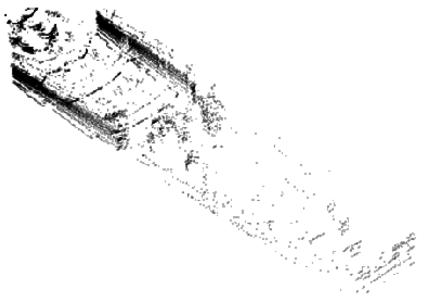

(b) Fake

Figure 5.10.: VAE - Range Map of a LiDAR scan in Seeing through fog dataset


(a) Corrupted Input

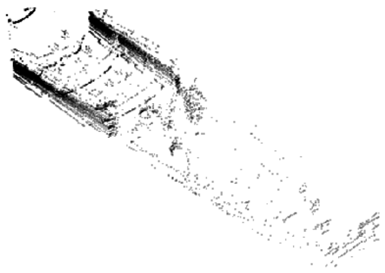
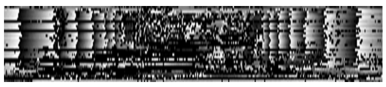

(b) Predicted denoised output

Figure 5.11.: WeatherNet - Point Clouds from Cerema datast

WeatherNet The WeatherNet model outputs the predicted semantic segment labels contains the classification of clear weather, fog, and rain. Using the predicted clear labels, the points that are referring to the clear labels are derived and plotted along with their corresponding input point cloud as seen in the figure 5.11. The input point that contains the weather-induced noise in the vicinity of LiDAR is denoised and it can be visibly seen in the filtered point cloud.

The range maps of both the input and the filtered LiDAR output are also shown in the figure 5.12. The weather-induced noise can be easily observed in the input range map and the corresponding denoised range map contains the LiDAR data where the noise is visibly removed.

Pix2Pix The weather-corrupted input images, the model generated fake images, and the corresponding clear weather images are analyzed by first reconstructing the range images to their corresponding point clouds representation and visually interpreting the results. And also, by visually interpreting the corresponding range maps.


(a) Corrupted Input

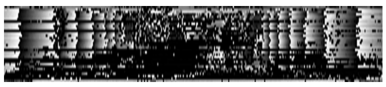

(b) Denoised Fake Output

Figure 5.12.: WeatherNet - Range Maps of Cerema datast

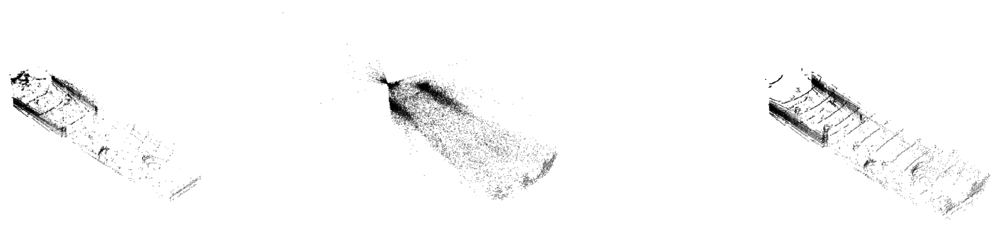
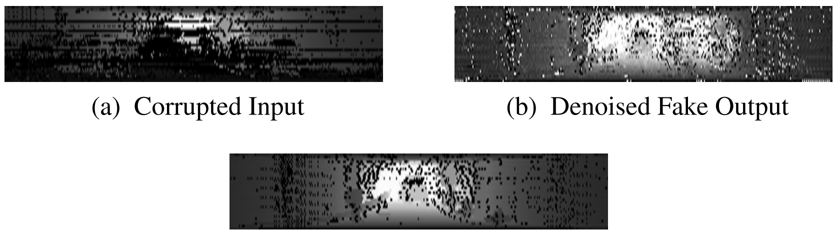

(a) Corrupted Input (b) Fake denoised output (c) Expected clear weather

Figure 5.13.: Pix2Pix - Point Clouds of Cerema dataset


(a) Corrupted Input (b) Denoised Fake Output

(c) Expected Clear weather

Figure 5.14.: Pix2Pix - Range Maps of Cerema dataset

As seen in the figure 5.13, the generate fake output doesn't yet generalize to the corresponding LiDAR structure when seen as a point cloud. The reason for this might be that the training data which is considered to be paired are not exactly paired data. The weather corrupted LiDAR data is paired with random clear-weather LiDAR data of the same scene and are not exact pairs. The range maps of the input generated image and the target clear weather image are shown in the figure 5.14. It can be seen that there is some amount of noise present in the vicinity of the LiDAR in the input image, however, in the generated fake range map, the noise has been reduced or even removed and the image is closer to the clear weather image. However, there is a certain amount of checkerboard artifacts in the generate fake range map.

MaskGAN The MaskGAN is evaluated for the two configurations of losses on both the Seeing-Through-Fog, and the Cerema dataset and the visualization of the sample point clouds can be seen in the figure 5.15. The sample (a, b) and (c, d) are the pairs containing the source and the fake generated point clouds by the MaskGAN trained on Seeing-Through-Fog dataset on simple loss and compound loss respectively. Similarly, the sample (e, f) and (g, h) are the pairs containing the source and the fake generated point clouds by the MaskGAN trained on Cerema dataset on simple and compound losses respectively.

The corresponding range map can be seen in the figure 5.16

On the Seeing-Through-Fog dataset, the visual results indicate that the MaskGAN, in both the configurations, performs similar to CycleGAN and tries to generate the point clouds closer to the target distribution. However, the generated point cloud scan lines do not completely adhere to the ones in an actual LiDAR scan. From the range map, we can see that the

Table 5.1.: FID between clear weather and clutter in datasets

	FID
Seeing through Fog	10.88

generated output is closer to the source input image but with artifacts.

However, on the Cerema dataset, as seen in the (e) - (f) and (g) - (h) of the figure 5.15 both the configurations of the model suffers to generate the point clouds preserving the integrity of the source point cloud. The corresponding fake generated range maps (f) and (h) in 5.16 also indicate the high amount artifacts present in the image.

U-Net The weather-corrupted input images, the model generated fake images, and the corresponding clear weather images are analyzed by first reconstructing the range images to their corresponding point clouds representation and visually interpreting the results. Also, by visual interpretation of the corresponding range maps.

As seen in the figure 5.17, the generate fake output doesn't generalize to the corresponding LiDAR structure when seen as a point cloud as well as in the range map 5.18.

5.3.2. Quantitative Analysis

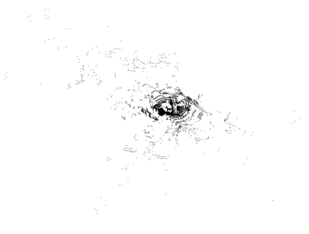
The quantitative results are obtained using some of the main metrics - EMD, CD, FID, RMSE - for the point cloud comparison as well as the new metric proposed in this thesis - RPOP. However, for the unsupervised models, only FID and RPOP metrics are shown, as they can provide a better comparison since the of the rest metrics are dependent on the availability of the ground truth. Supervised Models such as WeatherNet, Pix2Pix and U-Net were evaluated on EMD and FID.

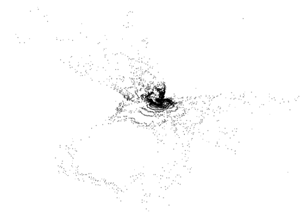
The unsupervised models are evaluated on the Seeing-Through-Fog datasets and the results can be seen in the table 5.2. As seen, MaskGAN with simple loss has a better FID score than the rest of the models, indicating it is more closer to the target clear weather distribution than the rest of the models

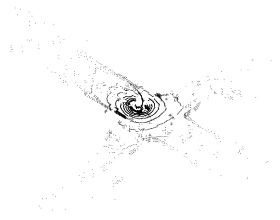
Table 5.3 shows the evaluations of the unsupervised models on Cerema dataset. All the models contain higher values of FID score indicating the gap between the generated and target image distributions. However, CycleGAN has a better FID score than the rest of the evaluated models

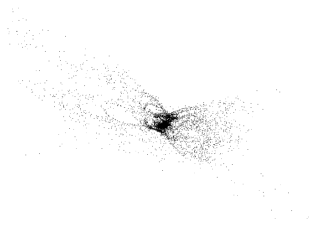
The evaluations on the supervised models can be seen in the table 5.4. Both the EMD and the FID score of the WeatherNet is better than the rest of the evaluated supervised models

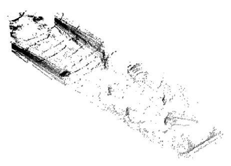
The values for the ratio of the percentage of points metric (RPOP) for the models trained on both the datasets can be seen in the table 5.5. It can be seen that in most cases, the clear RPOP remains closer to 1 indicating the clear points are almost the same in both source and the generated images, however, the background RPOP decreases and since the clear points remain the same, the fog and the rain points increases and hence the increase in the RPOP for both rain and the fog.

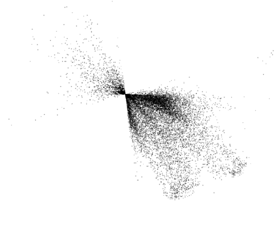

(a) Corrupted Input

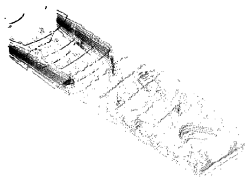

(b) Fake Output


(c) Corrupted Input


(d) Fake Output


(e) Corrupted Input


(f) Fake Output


(g) Corrupted Input

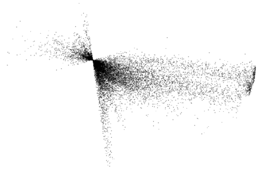
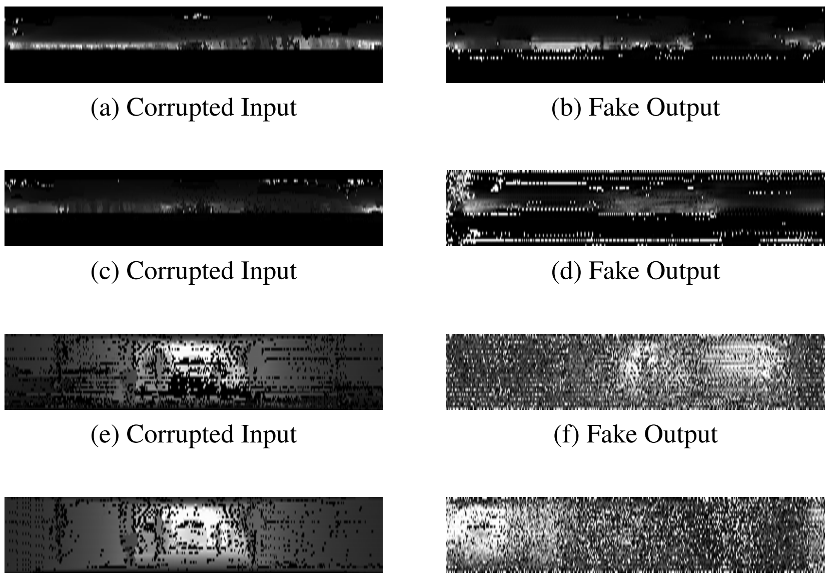

(h) Fake Output

Figure 5.15.: MaskGAN - Point clouds


(a) Corrupted Input (b) Fake Output

(c) Corrupted Input (d) Fake Output

(e) Corrupted Input (f) Fake Output

(g) Corrupted Input (h) Fake Output

Figure 5.16.: MaskGAN - Range Images

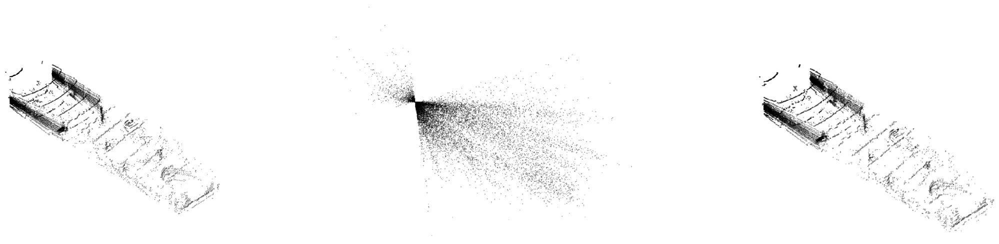
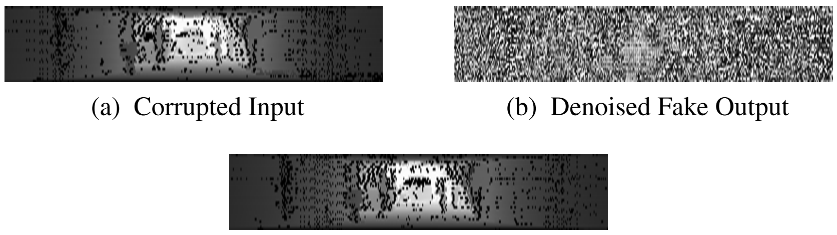

(a) Corrupted Input (b) Fake denoised output (c) Expected clear weather

Figure 5.17.: UNet based denoising - Point Clouds of Cerema datast


(a) Corrupted Input (b) Denoised Fake Output

(c) Expected Clear weather

Figure 5.18.: UNet based denoising - Range Maps of Cerema datast

Table 5.2.: Results on Seeing through fog dataset

Model	FID
VAE (Pre-trained on Kitti)	307
VAE	341.5
CycleGAN	166.6
MaskGAN (Simple loss)	142.6
MaskGAN (Compound loss)	238.33

Table 5.3.: Unsupervised model evaluations on Cerema dataset

Model	FID
CycleGAN	244.9
MaskGAN (Simple loss)	276.91
MaskGAN (Compound loss)	285.66

Table 5.4.: Supervised model evaluations on Cerema dataset

Model	EMD	FID
Weathernet	768	64.63
Pix2Pix	1158	209
Unet	1158	310.57

Table 5.5.: Results for the metric Ratio of Percentage of Point (RPOP)

Model	Background	Clear	Fog	Rain
VAE (STF)	0.91	0.99	1.18	0.80
VAE (Cerema)	4.56	0.34	1.87	2.91
CycleGAN (STF)	0.96	0.99	2.33	1.93
CycleGAN (Cerema)	0.23	0.45	16.44	0.56
MaskGAN (Simple loss - STF)	0.42	1.02	19	4.16
MaskGAN (Simple loss - Cerema)	0.24	1.02	4.27	2.70
MaskGAN (Compound loss - STF)	0.67	0.97	8.52	3.03
MaskGAN (Compound loss - Cerema)	0.60	0.38	25.3	1.65

6. Conclusion

In this thesis work, the general aspects involved in removing the weather-induced noise in the LiDAR point cloud and reconstructing the point clouds have been discussed in depth. In this line, several public LiDAR datasets, and also a private LiDAR dataset, have been explored and processed. Various representations of LiDAR data have been discussed and the methods to generate 2D range images from the 3D point cloud, and vice versa, have been developed. Various evaluation metrics were reviewed and new metrics, called the Percentage of Points (POP) and Ratio of Percentage of Points (RPOP) were developed as better indicators of the denoising and the reconstruction capabilities of the deep learning models. The datasets, such as the Seeing-Through-Fog and the Cerema, that contain weather-induced, corruptions have been used for training and evaluating the deep learning networks. Unsupervised image generation models, such as CycleGAN and VAE, were evaluated for its LiDAR range image denoising and reconstruction capabilities. However, the generative models, although can reconstruct the LiDAR range images, are inherently noise ignorant. On the other hand, a semantic-segmentation based LiDAR denoising model called the WeatherNet was implemented and evaluated on the Cerema dataset. This model is noise aware but lacks the reconstruction of the scattered point clouds. Hence, in this work, a novel method was proposed that combines the reconstructive behavior of the generative models and noise awareness of the WeatherNet model. This method was titled MaskGAN, and used a GAN along with an auxiliary pre-trained WeatherNet model, in which, masks that are predicted by the WeatherNet are used to get the non-corrupted points from the input data and compare with the non-corrupted points of the generated fake data. For comparison, two variants of loss function - Simple loss and Compound loss - were used. Both the variants of MaskGAN are trained on Seeing-Through-Fog and Cerema datasets and evaluated. Furthermore, to perform a supervised LiDAR range image translation, a pseudo paired dataset was created from the Cerema dataset and was first trained on Pix2Pix model with ResNet generator and evaluated. Later, a custom U-Net generator was used along with a custom loss function, containing the combination of TV loss, adversarial loss, and reconstruction loss and the model was evaluated.

The evaluation results indicate that, in most of the cases, there are checkerboard artifacts in the generated fake denoised images. This is attributed to the sparsity of the LiDAR point cloud and the fine-grained feature extraction capabilities of the convolution operators in the generators.

6.1. Future Work

With respect to the overall aspect of LiDAR point cloud denoising and the learning from evaluations of the previous models, there is a wide range of improvements that can be incorporated.

- Instead of the normal 2D Convolution operators in generators and discriminators, a sparse convolution operator to be used to extract the fine-grained features from the sparse point clouds
- Weather augmentations such as snow, rain, or fog augmentation on a clear LiDAR point cloud to be used. This paves way for a supervised learning approach to denoise and reconstructs the augmented LiDAR point clouds
- In MaskGAN, in place of an auxiliary WeatherNet model to remove the noise from the point cloud, the usage of a statistical filter, such as the DROR filter, can be explored.
- MaskGAN with advanced loss functions can be explored

A. Additional Evaluations

In this, the EMD, CD and RMSE values evaluated for the unsupervised models can be seen as in tables A.1 and A.2

Table A.1.: Results on Seeing through fog dataset

Model	EMD	CD	RMSE
VAE Pre-trained(Kitti)	668.79	0.003	0.0688
VAE	404.92	0.005909	0.06415
CycleGAN	392.55	0.0002	-
MaskGAN (Simple loss)	607.32	0.0008	-
MaskGAN (Compound loss)	447.41	-	-

Table A.2.: Results on Cerema dataset

Model	EMD
CycleGAN	587
MaskGAN (Simple loss)	1849
MaskGAN (Compound loss)	1738

B. Acronyms

LiDAR	Light Detection and Ranging
GAN	Generative Adversarial Network
AE	Autoencoders
VAE	Variational Autoencoders
BEV	Bird Eye View
PGM	Polar Grid Map
SOR	Statistical Outlier Removal
ROR	Radius Outlier Removal
DROR	Dynamic Radius Outlier Removal
EMD	Earth-Mover's Distance
CD	Chamfer's Discrepancy
RMSE	Root Mean Square Error
FID	Fréchet inception distance
IoU	Intersection Over Union
POP	Percentage of Points
RPOP	Ratio of Percentage of Points

List of Figures

List of Tables

Bibliography

[1] C. Jones, A. Smith and E. Roberts, "Article title," in *Proceedings Title*, vol. II. IEEE, 2003, pp. 803–806.

[2] I.-K. Lee, "Curve reconstruction from unorganized points," *Computer Aided Geometric Design*, vol. 17, no. 2, pp. 161–177, 2000. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167839699000448

[3] S. Fleishman, I. Drori and D. Cohen-Or, "Bilateral mesh denoising," *ACM Trans. Graph.*, vol. 22, no. 3, p. 950–953, Jul. 2003. [Online]. Available: https://doi.org/10.1145/882262.882368

[4] R. Roveri, A. Öztireli, I. Pandele and M. Gross, "Pointpronets: Consolidation of point clouds with convolutional neural networks," *Computer Graphics Forum*, vol. 37, pp. 87–99, 05 2018.

[5] P. Hermosilla, T. Ritschel and T. Ropinski, "Total denoising: Unsupervised learning of 3d point cloud cleaning," 2019.

[6] N. Charron, S. Phillips and S. L. Waslander, "De-noising of lidar point clouds corrupted by snowfall," in *2018 15th Conference on Computer and Robot Vision (CRV)*, 2018, pp. 254–261.

[7] R. Heinzler, F. Piewak, P. Schindler and W. Stork, "Cnn-based lidar point cloud de-noising in adverse weather," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, p. 2514–2521, Apr 2020. [Online]. Available: http://dx.doi.org/10.1109/LRA.2020.2972865

[8] A. E. Sallab, I. Sobh, M. Zahran and N. Essam, "Lidar sensor modeling and data augmentation with gans for autonomous driving," 2019.

[9] L. Caccia, H. van Hoof, A. Courville and J. Pineau, "Deep generative modeling of lidar data," 2019.

[10] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative adversarial networks," 2014.

[11] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2014.

[12] J.-Y. Zhu, T. Park, P. Isola and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," 2020.

[13] J. Digne and C. de Franchis, "The Bilateral Filter for Point Clouds," *Image Processing On Line*, vol. 7, pp. 278–287, 2017, https://doi.org/10.5201/ipol.2017.179.

[14] S. K. B. K., "Image denoising based on non-local means filter and its method noise thresholding," *Signal, Image and Video Processing*, vol. 7, pp. 1211–1227, 11 2013.

[15] H. Avron, A. Sharf, C. Greif and D. Cohen-Or, “<sub>1</sub>-sparse reconstruction of sharp point set surfaces,” vol. 29, no. 5, Nov. 2010. [Online]. Available: https://doi.org/10.1145/1857907.1857911

[16] X.-F. Han, J. Jin, M.-J. Wang, W. Jiang, L. Gao and L. Xiao, “A review of algorithms for filtering the 3d point cloud,” *Signal Processing: Image Communication*, vol. 57, 05 2017.

[17] O. Schall, A. Belyaev and H.-P. Seidel, “Robust Filtering of Noisy Scattered Point Data,” in *Eurographics Symposium on Point-Based Graphics (2005)*, M. Alexa, S. Rusinkiewicz, M. Pauly and M. Zwicker, Eds. The Eurographics Association, 2005.

[18] Y. Lipman, D. Cohen-Or, D. Levin and H. Tal-Ezer, “Parameterization-free projection for geometry reconstruction,” *ACM Trans. Graph.*, vol. 26, p. 22, 07 2007.

[19] C. R. Qi, H. Su, K. Mo and L. J. Guibas, “Pointnet: Deep learning on point sets for 3d classification and segmentation,” 2017.

[20] P. Guerrero, Y. Kleiman, M. Ovsjanikov and N. J. Mitra, “Pcpnetlearning local shape properties from raw point clouds,” *Computer Graphics Forum*, vol. 37, no. 2, p. 75–85, May 2018. [Online]. Available: http://dx.doi.org/10.1111/cgf.13343

[21] C. Duan, S. Chen and J. Kovacevic, “3d point cloud denoising via deep neural network based local surface estimation,” 2019.

[22] M.-J. Rakotosaona, V. L. Barbera, P. Guerrero, N. J. Mitra and M. Ovsjanikov, “Point-cleannet: Learning to denoise and remove outliers from dense point clouds,” 2019.

[23] S. Luo and W. Hu, “Differentiable manifold reconstruction for point cloud denoising,” *Proceedings of the 28th ACM International Conference on Multimedia*, Oct 2020. [Online]. Available: http://dx.doi.org/10.1145/3394171.3413727

[24] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*. Shanghai, China: IEEE, May 9-13 2011.

[25] F. Piewak, P. Pinggera, M. Schäfer, D. Peter, B. Schwarz, N. Schneider, D. Pfeiffer, M. Enzweiler and M. Zöllner, “Boosting lidar-based semantic labeling by cross-modal training data generation,” 2018.

[26] E. Denton, S. Chintala, A. Szlam and R. Fergus, “Deep generative image models using a laplacian pyramid of adversarial networks,” in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS’15. Cambridge, MA, USA: MIT Press, 2015, p. 1486–1494.

[27] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell and A. A. Efros, “Context encoders: Feature learning by inpainting,” 2016.

[28] J.-Y. Zhu, P. Krähenbühl, E. Shechtman and A. A. Efros, “Generative visual manipulation on the natural image manifold,” 2018.

[29] D. Bank, N. Koenigstein and R. Giryes, “Autoencoders,” 2021.

[30] D. P. Kingma and M. Welling, “An introduction to variational autoencoders,” *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, p. 307–392, 2019. [Online]. Available: http://dx.doi.org/10.1561/2200000056

[31] M.-Y. Liu, T. Breuel and J. Kautz, “Unsupervised image-to-image translation networks,” 2018.

[32] H.-Y. Lee, H.-Y. Tseng, J.-B. Huang, M. K. Singh and M.-H. Yang, “Diverse image-to-image translation via disentangled representations,” 2018.

[33] W. Wu, K. Cao, C. Li, C. Qian and C. C. Loy, “Transgaga: Geometry-aware unsupervised image-to-image translation,” 2019.

[34] Q. Yang, N. Li, Z. Zhao, X. Fan, E. I.-C. Chang and Y. Xu, “Mri cross-modality neuroimage-to-neuroimage translation,” 2018.

[35] T. Kim, M. Cha, H. Kim, J. K. Lee and J. Kim, “Learning to discover cross-domain relations with generative adversarial networks,” 2017.

[36] H.-Y. F. Tung, A. W. Harley, W. Seto and K. Fragkiadaki, “Adversarial inverse graphics networks: Learning 2d-to-3d lifting and image-to-image translation from unpaired supervision,” 2017.

[37] R. Zhang, J.-Y. Zhu, P. Isola, X. Geng, A. S. Lin, T. Yu and A. A. Efros, “Real-time user-guided image colorization with learned deep priors,” 2017.

[38] Y. Ding, X. Ma, M. Luo, A. Zheng and R. He, “Unsupervised contrastive photo-to-caricature translation based on auto-distortion,” 11 2020.

[39] Y. Yuan, S. Liu, J. Zhang, Y. Zhang, C. Dong and L. Lin, “Unsupervised image super-resolution using cycle-in-cycle generative adversarial networks,” 2018.

[40] Y. Pang, J. Lin, T. Qin and Z. Chen, “Image-to-image translation: Methods and applications,” 2021.

[41] P. Isola, J.-Y. Zhu, T. Zhou and A. A. Efros, “Image-to-image translation with conditional adversarial networks,” 2018.

[42] C. Wang, H. Zheng, Z. Yu, Z. Zheng, Z. Gu and B. Zheng, “Discriminative region proposal adversarial networks for high-quality image-to-image translation,” 2018.

[43] T.-C. Wang, M.-Y. Liu, J.-Y. Zhu, A. Tao, J. Kautz and B. Catanzaro, “High-resolution image synthesis and semantic manipulation with conditional gans,” 2018.

[44] H. Tang, D. Xu, N. Sebe, Y. Wang, J. J. Corso and Y. Yan, “Multi-channel attention selection gan with cascaded semantic guidance for cross-view image translation,” 2019.

[45] J.-Y. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang and E. Shechtman, “Toward multimodal image-to-image translation,” 2018.

[46] T. Park, M.-Y. Liu, T.-C. Wang and J.-Y. Zhu, “Semantic image synthesis with spatially-adaptive normalization,” 2019.

[47] A. Bansal, Y. Sheikh and D. Ramanan, “Pixelnn: Example-based image synthesis,” 2017.

[48] Z. Yi, H. Zhang, P. Tan and M. Gong, “Dualgan: Unsupervised dual learning for image-to-image translation,” 2018.

[49] T. Kim, M. Cha, H. Kim, J. K. Lee and J. Kim, “Learning to discover cross-domain relations with generative adversarial networks,” 2017.

[50] M.-Y. Liu, T. Breuel and J. Kautz, "Unsupervised image-to-image translation networks," 2018.

[51] M. Amodio and S. Krishnaswamy, "Travelgan: Image-to-image translation by transformation vector learning," 2019.

[52] S. Benaim and L. Wolf, "One-sided unsupervised domain mapping," 2017.

[53] Z. Shen, M. Huang, J. Shi, X. Xue and T. Huang, "Towards instance-level image-to-image translation," 2019.

[54] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang and J. Xiao, "3d shapenets: A deep representation for volumetric shapes," 2015.

[55] M. A. Uy, Q.-H. Pham, B.-S. Hua, D. T. Nguyen and S.-K. Yeung, "Revisiting point cloud classification: A new benchmark dataset and classification model on real-world data," 2019.

[56] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi and F. Yu, "Shapenet: An information-rich 3d model repository," 2015.

[57] I. Armeni, O. Sener, A. Zamir, H. Jiang, I. Brilakis, M. Fischer and S. Savarese, "3d semantic parsing of large-scale indoor spaces," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1534–1543, 2016.

[58] K. Mo, S. Zhu, A. X. Chang, L. Yi, S. Tripathi, L. J. Guibas and H. Su, "Partnet: A large-scale benchmark for fine-grained and hierarchical part-level 3d object understanding," 2018.

[59] T. Hackel, N. Savinov, L. Ladicky, J. D. Wegner, K. Schindler and M. Pollefeys, "Semantic3d.net: A new large-scale point cloud classification benchmark," 2017.

[60] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser and M. Nießner, "Scannet: Richly-annotated 3d reconstructions of indoor scenes," 2017.

[61] A. Geiger, P. Lenz and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[62] X. Song, P. Wang, D. Zhou, R. Zhu, C. Guan, Y. Dai, H. Su, H. Li and R. Yang, "Apollocar3d: A large 3d car instance understanding benchmark for autonomous driving," 2018.

[63] M. Bijelic, T. Gruber, F. Mannan, F. Kraus, W. Ritter, K. Dietmayer and F. Heide, "Seeing through fog without seeing fog: Deep multimodal sensor fusion in unseen adverse weather," in *The IEEE / CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[64] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine *et al.*, "Scalability in perception for autonomous driving: Waymo open dataset," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 2446–2454.

[65] Y. Guo, H. Wang, Q. Hu, H. Liu, L. Liu and M. Bennamoun, "Deep learning for 3d point clouds: A survey," 2020.

[66] H. Su, S. Maji, E. Kalogerakis and E. Learned-Miller, “Multi-view convolutional neural networks for 3d shape recognition,” 2015.

[67] T. Yu, J. Meng and J. Yuan, “Multi-view harmonized bilinear network for 3d object recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[68] C. R. Qi, H. Su, M. Niessner, A. Dai, M. Yan and L. J. Guibas, “Volumetric and multi-view cnns for object classification on 3d data,” 2016.

[69] Y. Feng, Z. Zhang, X. Zhao, R. Ji and Y. Gao, “Gvcnn: Group-view convolutional neural networks for 3d shape recognition,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 264–272.

[70] D. Maturana and S. Scherer, “Voxnet: A 3d convolutional neural network for real-time object recognition,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 922–928.

[71] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun and X. Tong, “O-cnn: Octree-based convolutional neural networks for 3d shape analysis,” vol. 36, no. 4, Jul. 2017. [Online]. Available: https://doi.org/10.1145/3072959.3073608

[72] C. R. Qi, L. Yi, H. Su and L. J. Guibas, “Pointnet++: Deep hierarchical feature learning on point sets in a metric space,” 2017.

[73] H. Zhao, L. Jiang, C.-W. Fu and J. Jia, “Pointweb: Enhancing local neighborhood features for point cloud processing,” in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 5560–5568.

[74] G. Qian, A. Abualshour, G. Li, A. Thabet and B. Ghanem, “Pu-gcn: Point cloud upsampling using graph convolutional networks,” 2021.

[75] Y. Liu, B. Fan, S. Xiang and C. Pan, “Relation-shape convolutional neural network for point cloud analysis,” 2019.

[76] P. Hermosilla, T. Ritschel, P.-P. Vázquez, Vinacua and T. Ropinski, “Monte carlo convolution for learning on non-uniformly sampled point clouds,” *ACM Transactions on Graphics*, vol. 37, no. 6, p. 1–12, Jan 2019. [Online]. Available: http://dx.doi.org/10.1145/3272127.3275110

[77] I. Seck, K. Dahmane, P. Duthon and G. Loosli, “Baselines and a datasheet for the cerema awp dataset,” 2018.

[78] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez and V. Koltun, “CARLA: An open urban driving simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[79] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan and O. Beijbom, “nuscenes: A multimodal dataset for autonomous driving,” 2020.

[80] C. B. Choy, D. Xu, J. Gwak, K. Chen and S. Savarese, “3d-r2n2: A unified approach for single and multi-view 3d object reconstruction,” 2016.

[81] G. Riegler, A. O. Ulusoy and A. Geiger, “Octnet: Learning deep 3d representations at high resolutions,” 2017.

[82] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun and X. Tong, "O-cnn," *ACM Transactions on Graphics*, vol. 36, no. 4, p. 1–11, Jul 2017. [Online]. Available: http://dx.doi.org/10.1145/3072959.3073608

[83] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," 2017.

[84] M. Tatarchenko, A. Dosovitskiy and T. Brox, "Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs," 2017.

[85] Z. Liu, H. Tang, Y. Lin and S. Han, "Point-voxel cnn for efficient 3d deep learning," 2019.

[86] B. Li, "3d fully convolutional network for vehicle detection in point cloud," 2017.

[87] L. Caltagirone, S. Scheidegger, L. Svensson and M. Wahde, "Fast lidar-based road detection using fully convolutional neural networks," 2017.

[88] A. Dewan, G. L. Oliveira and W. Burgard, "Deep semantic classification for 3d lidar data," 2017.

[89] M. Mirza and S. Osindero, "Conditional generative adversarial nets," 2014.

[90] J. Lehtinen, J. Munkberg, J. Hasselgren, S. Laine, T. Karras, M. Aittala and T. Aila, "Noise2noise: Learning image restoration without clean data," 2018.

[91] J. Batson and L. Royer, "Noise2self: Blind denoising by self-supervision," 2019.

[92] A. Krull, T.-O. Buchholz and F. Jug, "Noise2void - learning denoising from single noisy images," 2019.

[93] M. Javanmardi, M. Sajjadi, T. Liu and T. Tasdizen, "Unsupervised total variation loss for semi-supervised deep learning of semantic segmentation," 2018.

[94] Y. Rubner, C. Tomasi and L. J. Guibas, "The earth mover's distance as a metric for image retrieval." *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, 2000. [Online]. Available: https://doi.org/10.1023/A:1026543900054

[95] P. Achlioptas, O. Diamanti, I. Mitliagkas and L. Guibas, "Learning representations and generative models for 3D point clouds," in *Proceedings of the 35th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, J. Dy and A. Krause, Eds., vol. 80. PMLR, 10–15 Jul 2018, pp. 40–49. [Online]. Available: http://proceedings.mlr.press/v80/achlioptas18a.html

[96] H. Fan, H. Su and L. Guibas, "A point set generation network for 3d object reconstruction from a single image," 2016.

[97] P. Achlioptas, O. Diamanti, I. Mitliagkas and L. Guibas, "Learning representations and generative models for 3d point clouds," 2018.

[98] H. Fan, H. Su and L. J. Guibas, "A point set generation network for 3d object reconstruction from a single image," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[99] Q.-H. Pham, M. A. Uy, B.-S. Hua, D. T. Nguyen, G. Roig and S.-K. Yeung, "Lcd: Learned cross-domain descriptors for 2d-3d matching," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 11 856–11 864, 04 2020.

[100] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler and S. Hochreiter, “Gans trained by a two time-scale update rule converge to a local nash equilibrium,” 2018.

[101] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna, “Rethinking the inception architecture for computer vision,” 2015.

Declaration

Herewith, I declare that I have developed and written the enclosed thesis entirely by myself and that I have not used sources or means except those declared.

This thesis has not been submitted to any other authority to achieve an academic grading and has not been published elsewhere.

[illegible]

Stuttgart, 20.07.2021 Bharath Bangalore Somashekar