UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale in Advanced Computer Programming

Fault Detection in Autonomous Vehicle Perception Systems through Deep Learning

Anno Accademico 2023/2024

Relatore
Ch.mo prof. Luigi De Simone
Candidato
Giacomo Lisita
matr. M63001495

Abstract

Highly automated driving heavily relies on intricate machine learning systems, which are essential for interpreting vast amounts of data from various sensors to ensure safe and efficient vehicle operation. The accurate identification and understanding of sensor faults are critical to these systems, as they directly impact the reliability and performance of autonomous vehicles. Sensor faults can manifest as various abnormalities, ranging from unanticipated data values to alignment and obscuration errors caused by environmental factors or sensor malfunctions. These deviations, commonly referred to as "corner cases," represent rare but significant scenarios that challenge the perception systems of modern automobiles. Effectively handling these corner cases is crucial for building trust in the performance and safety of autonomous vehicles, especially those utilizing a diverse array of sensors, such as cameras and LiDAR, for environmental perception.

This research primarily focuses on advancing anomaly detection techniques with an emphasis on onboard camera and LiDAR sensors. It investigates the effectiveness of newly developed methodologies in dynamic driving contexts, where real-world conditions can vary widely and unpredictably. By integrating advanced computer vision, machine learning, and sensor fusion techniques, the study aims to enhance the ability of autonomous systems to detect and respond to anomalies in real time. Utilizing cutting-edge computer vision technologies, this approach underscores the development of robust anomaly detection algorithms capable of processing complex sensor data. The research ultimately seeks to pave the way for more reliable and efficient anomaly detection in future autonomous driving systems, contributing to the overall safety and advancement of automated vehicle technology.

Contents

Chapter 1

Understanding Anomalies in Sensor Systems

In the current era of technological evolution, the development and deployment of autonomous systems and vehicles has gained immense importance in various sectors. At the core of the operation of these autonomous entities are sensor systems, in particular *cameras* and *LiDAR* devices, that enable full environmental perception. The reliability, accuracy, and robustness of these sensors are considered the baseline to ensure the safety and effectiveness of autonomous operations. However, despite their advances, sensors are vulnerable to various forms of failure and anomalies resulting from environmental conditions, hardware malfunctions, or data processing errors. Unexpected and unknown situations often elude conventional visual perception methods. These scenarios, which are not observed during training, represent a significant obstacle as they can compromise road safety. Due to this problem, the detection of these cases is crucial and requires in-depth analysis of large data sets to identify the most suitable training examples. But in addition to enhancing data protection and selection, a trustworthy method for identifying these grey areas is crucial to gauging public opinion and acceptance of this cutting-edge technology. The development of advanced techniques that can identify, diagnose, and fix flaws is necessary.

The goal of this thesis is to investigate and further the use of deep learning(DL) methods in the field of sensor defect detection. Taking advantage of DL's innate capacity to identify confused connections in data, this study aims to develop resilient frameworks that can identify and minimize errors that can arise in these vital sensor systems.

The research question at the heart of anomaly detection in autonomous driving(AD) revolves around the effectiveness of new methodologies in identifying and responding to irregularities within

dynamic real-world environments. This includes understanding how state-of-the-art computer vision(CV) techniques, machine learning(ML) algorithms, and sensor fusion can collectively contribute to improving the ability of autonomous vehicles to detect and respond appropriately to anomalies.

Furthermore, the research seeks to explore the feasibility of integrating generative approaches to expand training datasets and improve the accuracy of anomaly identification in future AD systems. The approach adopted to identify anomalies in future AD, is quite comprehensive. Starting by collecting data from various urban and highway contexts from our own surveys and already existing datasets, covering a wide range of potential anomalies and driving scenarios. Once these data have been obtained, a careful pre-processing phase is then undertaken to ensure the data were ready to be elaborated on in the techniques considered. This phase involved various processes, including resizing all the images from different datasets to the same size and aspect size. After this phase, supervised learning methods are employed to classify the fault types, starting with a base architecture in which images are processed using only one CNN model that detects the presence of an error and to which class this error belongs. Subsequently, a two-tier approach is introduced, where the first model determines the presence of a fault and the second model identifies the specific fault injected. Furthermore, an image-to-image generation task has been performed to enhance and expand the original dataset starting from images in our dataset and the fault injected version of them. In addition, this research delves into the relationship between LiDAR and camera data, investigating how their integration can improve fault detection performance in autonomous systems. By employing these strategies, the thesis seeks to further the development of fault detection methods that are more trustworthy and efficient.

1.1 Statement of the Problem

Highly automated driving is fundamentally built on systems and functionalities that depend on ML. A crucial responsibility of these ML models involves accurately identifying and understanding atypical, unfamiliar, and possibly risky scenarios. Recognizing these scenarios, termed as *"corner cases"*, is of utmost importance for the effective creation, implementation, and verification of perception systems in upcoming vehicles, employing diverse sensor types.

The main purpose of this research topic is to inspect and detect various kinds of anomalies regarding the AD field, focusing on anomalies that could be found in cameras or LiDAR sensors.

In this thesis, we adhere to the categorization of corner cases by Breitenstein et al. [38], which classifies them into levels such as pixel, domain, object, scene, and scenario, each progressively

more challenging to identify. Building upon this, Heidecker et al. [27] expanded these levels from camera-based to include lidar and radar sensors. Consistent with their approach, we treat the "anomaly" and "corner case" as interchangeable terms. Within this study, our emphasis is on naturally occurring external corner cases.

1.2 What is an Anomaly?

The classification introduced in [10] defines anomalies within data as deviations from established norms, as vividly illustrated in 1.1 using a simple 2-dimensional dataset. The bulk of observations cluster within two defined regions, $N1$ and $N2$, defining the typical behavior. However, outliers such as points $o1$ and $o2$, along with the observations in region $O3$, stand apart significantly, signaling anomalies due to their considerable distance from the normative clusters.

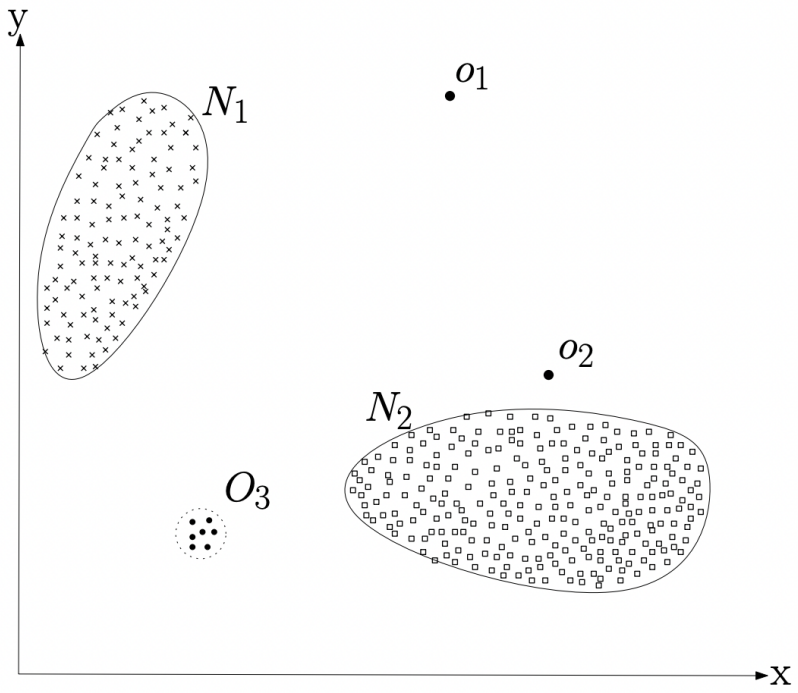

Figure 1.1: Simple example of anomalies in 2D dataset [10]

Real-world relevance or "interestingness" is a key property of an anomaly detection problem, as anomalies often contain information of critical value for analysis or decision-making. Although related, anomaly detection differs both from noise removal and noise accommodation. Noise removal involves eliminating unwanted data elements that are interfering with analysis; noise accommodation is what protects statistical model estimation from aberrant observations. A closely related concept also now coming to the fore is that of novelty detection: a way of identifying completely

new patterns in data, say new discussion topics in an online forum. The critical difference is that, once found, novel patterns tend to become part of the established norm, in contrast to anomalies, which persist outside this framework of what is normal.

Crucially, solutions developed for the related problems mentioned above often overlap with those for anomaly detection and are mutually applicable. This symbiosis motivates their inclusion in the review, with an acknowledgment of the fact that their methodologies in dealing with these challenges are closely related.

1.3 Different Labels for Anomalies

In literature, various terms are used to describe unexpected, or anomalous situations and events. In this section, we point out their distinctions and relations to corner cases in highly automated driving as stated in [27]. Various terms describe unexpected situations in highly automated driving.

Edge cases: Rare situations, often accounted for during development.

Corner cases: Result from combinations of normal situations, representing rare or unconsidered cases.

Outliers: Deviating observations possibly generated by different mechanisms.

Anomalies: Deviations from learned patterns or general behavior.

Novelties: Unseen instances or changes in known processes.

1.4 How does an anomaly present itself?

A comprehensive framework for categorizing and understanding corner cases within the visual perception systems of automated driving is introduced in [27] like a systematic breakdown of these cases into distinct levels, each representing varying complexities in detection.

Beginning with *pixel-level* anomalies, such as overexposure or dead pixels, it progresses to domain-level shifts caused by changes in location, weather conditions, or time of day. *Object-level* corner cases involve singular anomalies or novelties, like encountering wild animals on the road or unusual objects such as walking aids like a rollator or crutches.

scene-level anomalies differentiate between contextual anomalies (known objects in unexpected locations, like a tree obstructing a street) and collective anomalies (known objects appearing in unusual quantities, such as a demonstration).

At the apex of complexity lie *scenario-level* corner cases, observed over a sequence of images, encompassing risky situations like overtaking a cyclist, novel scenarios without observed occurrences but not necessarily leading to collisions, and anomalous scenarios presenting a high potential for collisions, like a person suddenly walking into the street in front of the autonomous vehicle. In the table 1.1 there is a graphical representation of this classification proposed by Breitenstein et al. [38].

Table 1.1: Summary of Corner Case Levels in Automated Driving Visual Perception [38]

Corner Case Level	**Description**
Pixel Level	Global outliers (e.g., overexposure) and local outliers (e.g., dead pixels)
Domain Level	Shifts in domain (e.g., location, weather, time of day)
Object Level	Single-point anomalies (e.g., wild animals, uncommon objects on the road)
Scene Level	Contextual anomalies (known objects in unusual locations) and collective anomalies (known objects in unusual quantities)
Scenario Level	Risks observed over image sequences (e.g., overtaking a cyclist), novel scenarios (not observed but non-collision potential), and anomalous scenarios (not observed but high collision potential)

In this work we will focus on anomalies that could be found in the domain and pixel level given that the most part of the sensor faults are related to these levels.

1.5 Sensor Faults and Failures

The functionality and importance of cameras and LiDAR sensors in contemporary perception systems underscore their critical role in advancing modern technology. Cameras provide rich visual information, capturing color, texture, and fine details, while LiDAR sensors offer precise depth measurements and 3D mapping capabilities. By fusing camera and LiDAR data, the strengths of each sensor are not only enhanced but also combined to create a more comprehensive and accurate understanding of the environment. This fusion effectively compensates for the limitations inherent in individual sensors, such as a camera's sensitivity to lighting conditions or LiDAR's challenges with reflective surfaces.

This multi-sensory approach forms the bedrock of robust perception systems, ensuring that these technologies can operate reliably in a wide range of conditions and environments. The enhanced reliability and adaptability provided by this integration are crucial for diverse real-world applications, from autonomous vehicles navigating complex urban landscapes to drones performing precise mapping and inspection tasks.

1.5.1 Applications and Challenges

Numerous sectors have been revolutionized by the applications of cameras and LiDAR sensors, which have contributed to advancements in perception-based technology. These sensors are the main source of environmental data for autonomous cars, allowing for functions like obstacle avoidance, lane tracking, and object detection. While LiDAR provides the accurate distance measurements required to map the vehicle's surroundings in three dimensions, cameras provide the visual signals necessary for recognizing road conditions, traffic signs, and pedestrian movements. Furthermore, the combination of camera and LiDAR data in robotics makes it easier to navigate, manipulate objects, and map environments. This increases the autonomy and adaptability of robotic systems in a variety of industries, such as manufacturing, healthcare, and logistics.

But these sensors deal with many difficulties such as environmental factors that can affect cameras, for example changes in lighting, occlusions, and bad weather, which can affect the quality of the images and the accuracy of object detection. LiDAR systems are good at gathering accurate spatial data, but they have trouble with reflecting surfaces, repeated returns from intricate geometries, and objects that move quickly. Furthermore, both sensors are vulnerable to data processing complications, calibration problems, and hardware failures. By resolving these issues, perception systems' robustness and dependability will be improved, and their full potential will be realized in a variety of applications.

Chapter 2

Current State of Research in Autonomous Driving Anomaly Detection

Within the scope of AD, corner cases exist at multiple levels, involving diverse sensor technologies such as cameras, LiDAR, and radar in [14] we can find an overview of the main researches. This study provides an overview of anomaly detection methods within the domain of AD across various sensor modalities. We encompass methods not explicitly developed for AD but deemed applicable. These techniques are characterized across different modalities, including the camera, LiDAR, radar, multimodal, and abstract object levels. The characterization involves their general detection approach, corner case type, evaluation dataset or simulation, and potential online application.

Detection methods are categorized into five concepts, following the framework by Breitenstein et al.: "reconstruction, prediction, generative, confidence scores, and feature extraction" [38]. *Confidence score* techniques, often derived through post-processing without affecting the neural network(NN)'s training, are further classified into Bayesian approaches, learned scores, and scores obtained through post-processing. *Reconstruction approaches* aim to restore normality, considering any deviation as anomalous. *Generative approaches* closely align with reconstruction methods but also incorporate the discriminator's decision or the proximity to the training data. *Feature extraction* involves utilizing handcrafted or learned features to determine a class label or compare modalities at various feature levels. *Prediction-based techniques* forecast the next expected frame(s) under normal conditions. In this study, is conducted an extensive examination of methods for iden-

tifying irregularities in the sphere of AD, the results are represented in 2.1. Presently, the majority of recent advancements primarily revolve around anomaly detection in image-based systems.

However, LiDAR and radar-based methods are encountering challenges in gaining traction. One prominent reason for this discrepancy is the absence of standardized benchmarks, which, up to now, exist only in the domain of camera-based systems. This lack of universally accepted datasets containing labeled anomalies complicates the unified comparison of detection techniques. For instance, LiDAR-based techniques notably focus on identifying single-point anomalies. As an overview, the current state-of-the-art methodologies particularly excel in detecting contextual anomalies at the scene level, while their capability in identifying collective anomalies is relatively underdeveloped.

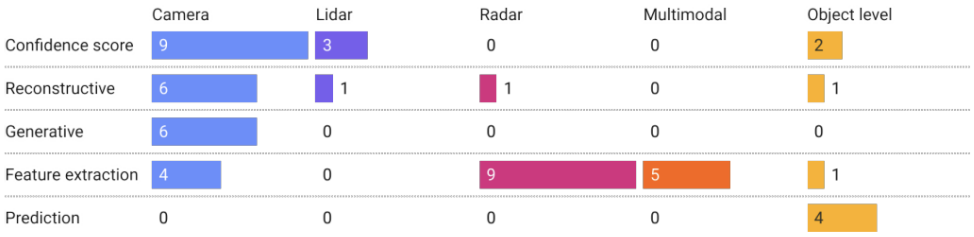

Figure 2.1: Overview of the analyzed studies [14]

2.1 Studies Utilizing Camera Data

Although these studies primarily focus on anomaly detection, which is outside the scope of this thesis, it is still worthwhile to review the state of the art in anomaly detection more broadly, given our focus on fault detection.

The main focus of this literature review will be on reconstructive, generative and feature extraction approaches. But, some of the main approaches utilizing confidence scores in NNs serve as a fundamental framework for anomaly detection, encompassing diverse methodologies to identify deviations or irregularities within data. Kendall et al.'s Bayesian SegNet [34] pioneered this by determining SegNet's uncertainty in semantic segmentation (SemSeg) using Monte Carlo dropout sampling. Variance among classes signifies varying degrees of uncertainty, allowing interpretation of this uncertainty as pixel-wise anomaly scores for road obstacle detection.

Jung et al. [33] similarly aim to detect unknown road obstacles, employing class-conditioned standardized max logits from a segmentation network. This method is motivated by the diverse ranges that max logits exhibit across predicted classes. They derive mean and standard deviations from training samples, categorizing this standardization as a confidence score approach. Addi-

tionally, they mitigate class boundaries, applying dilated smoothing to encompass local semantics within broad receptive fields.

Bevandic et al. [2] introduce a multi-task network that simultaneously segments input frames into semantics while generating an anomaly probability map. This map overrides SemSeg when a probability exceeds a threshold, calibrating the confidence score when the model encounters outliers.

More recently, Du et al. [17] presented the *Virtual Outlier Synthesis* (VOS) learning framework, shaping NN decision boundaries by synthesizing virtual outliers. Initially, they estimate a class-conditioned multivariate Gaussian distribution in the penultimate latent space. Subsequently, outliers are sampled from a sufficiently small ϵ-likelihood region of this learned distribution. These virtual outliers near the class boundary encourage the model to establish a compact decision boundary between in-distribution (ID) and out-of-distribution (OOD) data. They propose a novel training objective using free energy as an uncertainty measurement, assigning negative energy to ID data and positive energy to virtual outliers. During inference, OOD objects are detected using a logistic regressor based on the uncertainty score.

Prior approaches primarily focus on object-level anomalies, whereas Breitenstein et al. [5] pioneered collective anomaly detection. They learn normal quantities of class-instances from a reference dataset, predicting class-instances via Mask R-CNN to derive a discrete class distribution. Additionally, they introduce a variation of the earth-mover's distance (EMD) for inference, termed the earth-mover's deviation (EMDEV). EMDEV provides also a signed value indicating whether a scene contains more or fewer instances of a class than usual.

2.1.1 Reconstructive and generative approaches

Anomaly detection is using a combination of reconstructive and generative approaches to guarantee robustness and safety in an AD context. In most fields, reconstructive strategies aim to reproduce or rebuild what is normative or standard; deviation from this standard is labeled as abnormal or deviant. These approaches attempt to rebuild a standard or expected model, trying to bring deviant behavior back into line with this established norm. While these generative approaches share similarities with the reconstructive methods mentioned earlier, they proceed to do more than the reconstruction itself.

One of the first attempts to address the problem of unknown objects in semantic segmentation is by Lis et al.'s [41], which predict the probability of pixels belonging to unseen classes. While more conventional semantic segmentation approaches only take into account the known classes

during training, this method could counter the presence of the most unexpected objects. Here, a segmentation algorithm is used to produce a map, which, when passed through a generative network, allows a reconstruction of the input image. Objects that do not fall under the applicability of the segmentation algorithm are detected based on discrepancies of the original image and the generated one. To perform this, a dedicated "discrepancy network" is trained using co-segmentation technique as a reference. The network uses a developed three-stream architecture and Visual Geometry Group(VGG) networks combined with proprietary CNNs for predicted labels processing and feature correlations computation, presented in figure 2.2. Here, based on the differences of the original and generated images, the network can adequately specify which of the inconsistencies are meaningful.

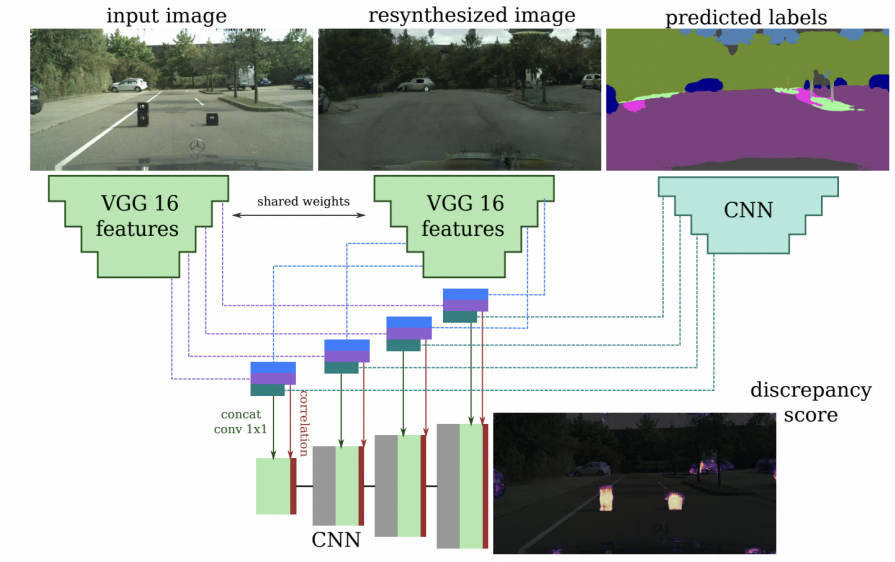

Figure 2.2: Discrepancy network proposed by Lis et al's. [41] using VGG16 and CNNs. The input image is processed to generate resynthesized images and predicted labels, then, a discrepancy score is computed through CNNs, highlighting potential anomalies by comparing these outputs.

A detection framework is proposed for the segmentation of anomalous instances, drawing inspiration from recent re-synthesis approaches and expanding upon them by incorporating the advantages of uncertainty estimation methods. The framework is introduced along with its respective modules in figure 2.3. In the final step, the framework's output is amalgamated with the calculated uncertainty maps to create a cohesive ensemble method. This method aims to mitigate false positives and diminish overconfidence in the predictions, resulting in an enhanced anomaly detection system. The system is composed of three main modules:

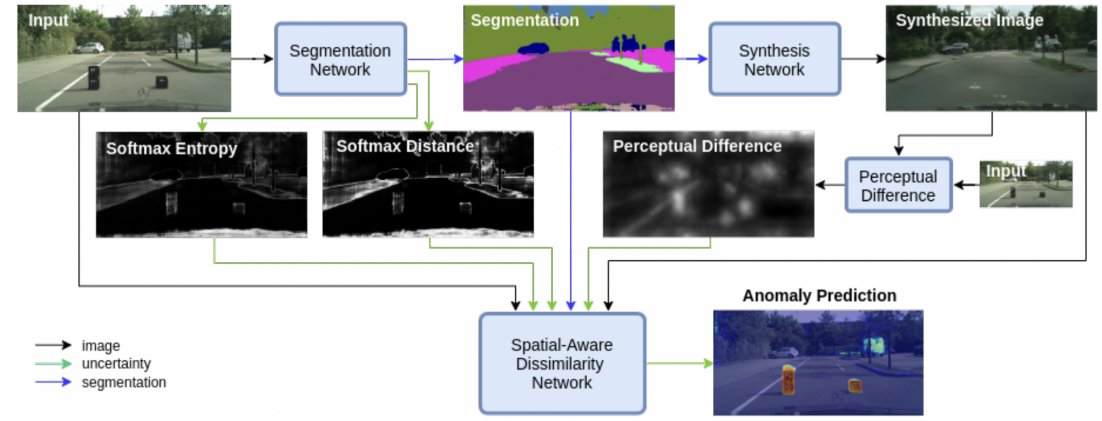

Figure 2.3: Architecture of network proposed by Di Biase et al's. [15]. The framework integrates a segmentation network and a synthesis network to generate segmentation maps and synthesized images. Key metrics such as Softmax entropy, Softmax distance, and perceptual difference are computed and fed into a spatial-aware dissimilarity network to predict anomalies in the input image.

- **Segmentation Module:** Employs a segmentation network to generate a semantic map from the input image. Additionally, it calculates uncertainty through dispersion measures like softmax entropy and softmax distance.
- **Synthesis Module:** Utilizes a conditional generative adversarial network (cGAN) to create realistic images based on the semantic map. Introduces a perceptual difference measure to compare feature representations between the original and synthesized images.
- **Dissimilarity Module:** Combines original and generated images, semantic maps, and uncertainty measures to predict anomaly segmentation maps. Includes an encoder (using pretrained VGG networks and CNNs), a fusion module (employing convolution and correlation operations), and a decoder for the final anomaly segmentation prediction.

The model combines measures of uncertainty and perceptual differences to improve anomaly detection, focusing on achieving the detection of differences between the synthetic and original images.

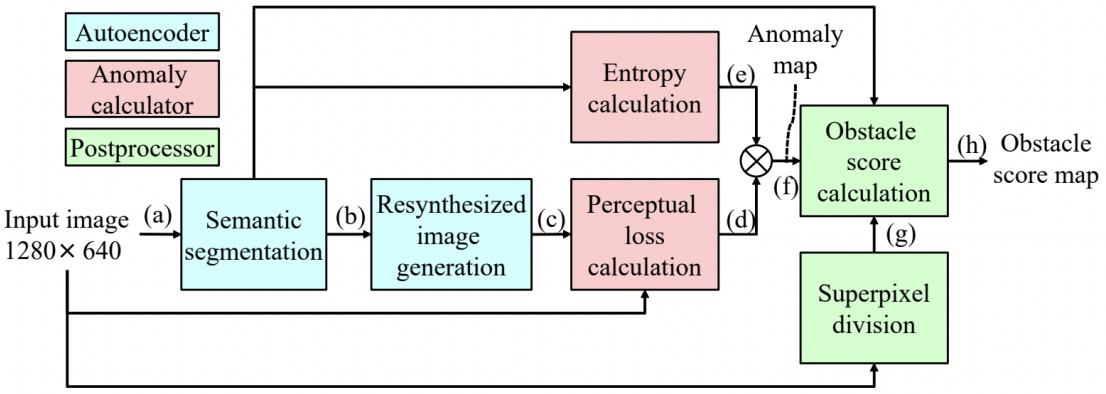

Figure 2.4: Autoencoder, Anomaly calculator and Postprocessor [47]. The architecture undergoes semantic segmentation and resynthesized image generation, followed by entropy and perceptual loss calculations. The outputs are combined to generate an anomaly map, which is further processed with superpixel division to calculate an obstacle score map, identifying and scoring potential obstacles in the scene.

Another work in this line is presented in [47]. The current model is conceptually aligned with the earlier models, however, the implementation of this model is significantly different and stands as one of the highly rational and novel implementations in this domain. In this approach, there is a three-stage model that includes an *autoencoder*, an *anomaly calculator*, and a *postprocessor*. In the first stage, the autoencoder is used with the help of a regular in-vehicle camera, which has a color image as input. The model was then used to develop a semantic map through a method that produced semantic segmentation, which was then followed by the production of a resynthesized image through photographic image synthesis. The anomaly calculator was then developed using an improvement upon this model. In this case, the anomaly map was generated using a perceptual loss and entropy used by the semantic map. Finally, the postprocessor sharpens the anomaly information at local levels and results in the obstacle score map. Each of the steps and a representation thereof are described in detail below to have better insight into this model.

- **Autoencoder:** The following autoencoder is based on the introduction of important modules related to semantic segmentation and resynthesized image generation. This block generates the semantic map and the corresponding resynthesized image, sending these to the anomaly calculator and postprocessor. This is done using a broadly used semantic segmentation, which is taken from the Cityscapes dataset. The Cityscapes dataset is an important dataset that contains road scenes. The dimensions of the dataset images are brought down to 1,280 × 640 pixels to fit the dimensions for GPU memory. The semantic segmentation and the resynthesized image generation are concatenated using advanced semantic segmentation parameters, which are based on the technique cascaded refinement network [12], for the formation of the

Cityscapes [13] dataset input and the resynthesized image. It is important to note that the autoencoder is the only component that needs to learn the model parameters among the three main components. Importantly, this autoencoder has the added ability to enhance the quality of resynthesized images with its lightweight deep NNs linked by changing the connection of the decoder to the intermediate layer instead of linking to the last layer. This means other features like instance segmentation and instance-level feature embedding are not required, which would be needed to train the functions according to the requirement as per the *Pix2PixHD* [62]. Also, the encoder and decoder have to be trained separately, as in *Resynth*, but in this autoencoder, end-to-end learning is done, and lightweight deep NNs are concatenated for faster inference.

- **Anomaly calculator:** It consists of complex modules for both entropy and perceptual loss calculation. Its main responsibility is to produce an anomaly map; in other words, it attributes an anomaly score to every pixel in the input image, and then sends this anomaly map to the postprocessor. In this process, some assumptions are made in calculating the semantic labels of an unknown object. First of all, the semantic map will be very ambiguous around this object, as a result, since there will be ambiguity, there would be significant dissimilarities between the resynthesized and input images in appearance. In a nutshell, entropy is calculated for the semantic map for ambiguousness, while the perceptual loss gives a measure of the discrepancies in appearance. Both metrics are combined to give the anomaly score. In other words, the calculation of the entropy of the semantic map by formulating the probability estimation of the semantic labels using the semantic segmentation technique is demonstrated, together with an upconverter, to match the resolution of the resynthesized image to the input image resolution. Additionally, the perceptual loss is demonstrated as the evaluation of the discrepancies in multiple VGG19 layers between the input and resynthesized images. As a result, the anomaly map is calculated with the element-wise multiplication of these measures to combine the perceptual loss and entropy to determine the anomalies in the image.

- **Postprocessor:** Composed by two modules, these modules are used to calculate the obstacle score and division of superpixels, which are necessary to locate the unidentified objects in the input image. Given an input image, it is segmented into localized regions called superpixels. This module has been realized by a simple linear iterative clustering algorithm. The reason why this segmentation method is used is that it can correctly represent the input image and clusters the input image into distinguishable regions without breaking objects. After the

input image segmented into superpixels, the postprocessor module calculates the obstacle score of all the superpixels. The equation of calculating the obstacle score is given based on parameters such as anomaly score average, number of pixels, the average probability value of the road label, and the distance between superpixels center. The postprocessor module checks the above parameters and gives the result based on the calculated obstacle score is greater than or lesser than the fixed threshold. It shows that the superpixel region contains or does not contain a road obstacle.

The method proposed in [61] centers on acquiring a compact, robust latent representation of road surfaces, leveraging datasets rich in labeled road information, distinct from anomalous objects. By harnessing this latent road representation, the approach conducts road reconstruction in tandem with widely available semantic segmentation networks, such as those used in autonomous vehicles, fostering robust anomaly detection on roads. This adaptable principle extends beyond road scenarios, offering potential applications like anomaly detection on water surfaces using naval drones. The devised deep NN model concurrently learns pixel-level road reconstruction and integrates it with semantic segmentation. Trained on Cityscapes dataset, the road reconstruction coupled with semantic segmentation exhibits robust generalization across multiple anomaly detection datasets, crucial for tasks where comprehensive training data is scarce. The architectural framework, elucidated in Fig. 2.5, comprises two central modules – reconstruction and segmentation coupling – alongside a straightforward yet efficient data augmentation strategy.

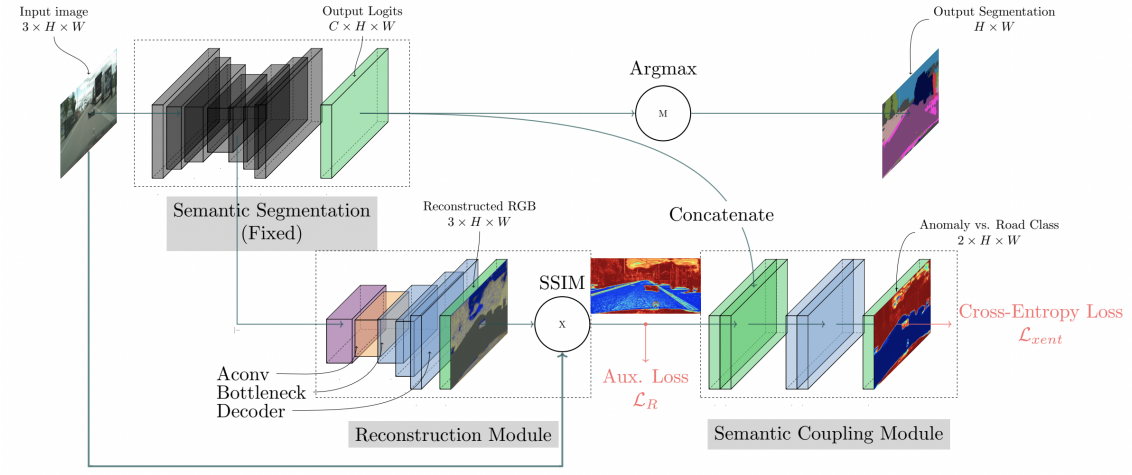

Figure 2.5: Architectural framework depicting the integration of segmentation coupling and reconstruction modules for anomaly detection [61]. The input image is segmented, and reconstructed RGB features are compared using SSIM. These features are then concatenated in the semantic coupling module to detect anomalies and differentiate road classes, with optimization through cross-entropy and auxiliary loss functions.

- **Reconstruction module:** The module in this approach aims to discern the appearance of drivable road surfaces in a discriminative manner, minimizing errors in road reconstruction while allowing substantial errors in other environmental elements. A deep NN, acting as a decoder, connects to a pre-trained segmentation network's backbone. This configuration, employing discriminative reconstruction loss with a small bottleneck, facilitates anomaly detection by identifying poorly reconstructed regions in the error image. By utilizing a fixed backbone, the reconstruction module can integrate with existing semantic segmentation networks, necessitating training only on the Reconstruction Module and segmentation coupling. The module comprises three core components:
  - reduction of backbone features' dimensionality for the decoder bottleneck, made via atrous spatial pyramid pooling (ASPP), enabling multi-scale appearance capture
  - a decoder upsampling feature channels to reconstruct the road, employing convolutional blocks with operations such as bilinear upsampling, 2D convolution, batch normalization, and ReLU nonlinearity
  - comparison of the reconstructed RGB image to the input image using the structural similarity index measure (SSIM), albeit noting its neighborhood-averaged nature might not perfectly align with human perception when employed as a loss function in backpropagation. The study includes an ablation analysis, contrasting the SSIM with the widely used L2 norm for per-pixel image reconstruction accuracy.
- **Segmentation Coupling Module:** serves to merge information from the "known classes" inferred by the fixed segmentation network's output logits with the "unknown anomalies" identified by the reconstruction module as poorly reconstructed image regions. This fusion is crucial, given the tendency of segmentation networks to exhibit overconfidence in class likelihood estimation, leading to misclassification of small anomalous road objects. To facilitate the combination of semantic segmentation and reconstruction data, a standard convolution block is proposed. This block concatenates segmentation logits with SSIM reconstruction errors, passing them through two convolutional blocks followed by batch normalization and ReLU non-linearity. The output, encompassing anomaly and road class channels, undergoes normalization via a softmax layer, and training employs a standard binary cross-entropy loss. The overall loss function aggregates the binary cross-entropy loss with a scaled auxiliary reconstruction loss, creating a unified loss without explicit weighting between the two, considering their similar scale under normal conditions.

The aim of the study [39] is to identify any obstacles in the drivable space, important for

trajectory planning in self-driving systems. These obstacles can be potentially hazardous objects that are, however, located within the area that the previous stage of the autonomous vehicle's perception pipeline considers drivable. The detection of such obstacles presents challenges because of the different forms they can take and their potential absence from the database used to train object recognition networks. Their method aims to find such unexpected obstacles that appear in the estimated road region, not defined explicitly or in any form of exemplars. They provide a two-stage methodology: erasing obstacles by inpainting road patches in a sliding-window manner and then using a discrepancy network to compare the original image with the inpainted version to determine if they match well enough. The intuition for this lies in the fact that the regions with obstacles will obviously have large differences when inpainting, but the regions without any obstacles will show some similarities but will not be equivalent. The disparity network uses this disparity to estimate a discrepancy. It does so by generating a heat map that shows the probability of a pixel in the drivable area belonging to an obstacle.

2.1.2 Feature extraction

The application of feature extraction methods for addressing domain shift detection has been extensively explored across a range of studies, each employing different approaches to tackle this challenging issue.

Zhang et al. ([70]) developed the **DeepRoad** framework, which specifically addresses domain shift by computing the distance between individual input images and the training embeddings derived from VGGNet features. This method provides a robust way to quantify the difference between the input data and the learned representation, enabling effective detection of domain shifts that might occur due to changes in environmental conditions or sensor configurations.

Bolte et al. (as discussed in [4]) proposed a different approach, which focuses on calculating the mean-squared error of feature maps. Their technique involves computing the mean-squared error across an entire dataset or a batch, which serves as a metric for detecting domain shifts. This approach leverages the consistency of feature maps across different domains, allowing the identification of discrepancies that indicate a shift.

Bai et al. ([1]) targeted the detection of anomalies in urban road scenes, a specific yet critical aspect of domain shift detection. Their approach involves classifying complete input scenes into an anomalous category by establishing representatives for normal urban scenes. This is achieved by clustering scale-invariant feature transform (SIFT) features using k-means clustering. The resulting clusters are then used to classify new images, utilizing a one-class support vector machine (one-class SVM) to distinguish between normal and anomalous scenes. This method effectively captures the inherent variability of urban environments while focusing on deviations that indicate potential domain shifts.

Many of these techniques are designed to be self-sufficient, operating independently without relying on external data sources. This self-sufficiency is a significant advantage, as it allows the methods to function effectively in a wide range of scenarios without the need for additional data inputs. However, one common challenge with these methods is the need for adaptation. Typically, adaptation involves retraining either the proposed extension module or, in some cases, the entire detection architecture. This retraining is necessary to ensure that the model remains accurate and effective as it encounters new domains or experiences shifts in the data distribution.

2.2 Studies Utilizing LiDAR Data

Autonomous vehicles rely on a combination of sensors to be able to perceive their environment. These cameras provide high fidelity images but it is difficult to estimate depth from them. In

order to bridge this deficit, LiDAR sensors are used simultaneously in order to build 3D maps and give information about depth. While there is a lot of work being done on LiDAR data to enhance the quality of the data that are obtained with high resolution, what we are interested in, is the detection of anomalies in a global scope. Thus, what we aim at is finding anomalies not only in individual points but in whole clusters or wherever the change in appearance is severe and consistent. Here, the weather, such as rain, snow and fog significantly affect the quality of the data. These methods go further in anomaly detection than in other points but in larger groups or noticeable shifts in appearance, which also helps understand how weather conditions can affect the analysis of the data using LiDAR.

2.2.1 Confidence score

The research [68] depends on this aim but focuses on rainy weather condition. Recent studies have discussed the interaction between LiDAR technology and rain, elaborating three interactions that cause degradation: *absorption*, *reflection*, and *refraction*. Each of these interactions has different effects on degradation.

Absorption might reduce the laser beam energy that, in turn, might impact the number of points returned, the maximum detection range, and the LiDAR point intensity. If the laser beam reflects from raindrops rather than solid objects, noisiness might appear in LiDAR scans.

Reflection is the case in which the laser beam meets raindrops and reflects on their surfaces. This can produce great noise in the LiDAR data as the beam is scattered in different directions, which produces false readings and ghost points. This scattering effect reduces the accuracy and reliability of the data, since the LiDAR system might interpret raindrops as solid objects, leading to errors in mapping and detecting obstacles.

Refraction occurs whenever the laser beam penetrates a raindrop and affects the accuracy of the laser measurement.

The key method for degradation quantification consists of steps of *LiDAR-image conversion* and *Degradation extraction and quantification*. In the first step LiDAR the 3D LiDAR scans are converted into 2D LiDAR images by projecting them spherically. The dimensions of the conversion are determined by the number of laser channels and LiDAR angular resolution. As a result, an image is generated where each pixel corresponds to a laser beam, and the intensity values represent the received LiDAR points. The motivation behind converting 3D LiDAR measurements into 2D images is to preserve the degradation information caused by rain. Changes in the values of intensity indicate the presence of degradation. The subsequent step, which involves the extraction

and quantification of degradation, aims to detect and measure the extent of degradation in LiDAR images.

The *DeepSAD* approach [53] is used when dealing with datasets labeled as rainy and normal. The objective is to determine a measure of degradation in a given test image by considering both labeled (rainy and normal) and unlabeled LiDAR image datasets. DeepSAD employs a neural network to learn a latent representation of normal input images, which are mapped to the center of a hypersphere. In contrast, rainy input images are mapped further from this center, indicating a higher level of degradation.

A test image's degradation score is defined as the distance between its mapping in the learned latent space and the predetermined hypersphere center. The objective is to capture and quantify various degradations caused by rain, making this method a robust approach for assessing LiDAR image quality in adverse weather conditions.

2.2.2 Closed-set settings

As point cloud object identification has evolved, a number of architectural frameworks have become important participants in the industry. Among them, *VoxelNet* [71], *PointRCNN* [55], and *PointNet++* [51] stand out since its main objective is to precisely recognize and categorize objects in point cloud data. But a common shortcoming among these designs is that they operate in a closed-set scenario, meaning that they are limited to identifying and classifying object classes that are explicitly present in their training datasets.

To go into more detail on the *closed-set* setting, this basically means that these object detection models can only recognize and categorize objects that they have come across in the training process. Any object that doesn't fit into one of these pre-established classifications is probably going to be incorrectly classed or ignored. One major disadvantage of the closed-set paradigm is its rigidity, which can be especially problematic in dynamic contexts where the diversity of items may be greater than what was included in the initial training data. This departure from the closed-set approach liberates the model from the constraint of assigning each detection to one of the predetermined classes. Consequently, this flexibility is expected to lead to an improvement in the false positive rate, as the model becomes more adaptive to recognizing and acknowledging the presence of novel objects that were not part of its training regimen. To elaborate further, the open-set setting allows the model to embrace the novelty of objects by treating them as unfamiliar instances rather than attempting to force them into predefined categories. This adaptability is particularly valuable in scenarios where the environment introduces new, previously unseen object

classes. By acknowledging the potential existence of unknown objects, the model gains a more nuanced understanding of its surroundings and becomes better equipped to handle diverse and evolving datasets.

2.2.3 Open-set settings

Open-set detection techniques have introduced a paradigm change in response to this constraint. In contrast to their closed-set equivalents, these methods are intended to specifically identify and label objects that do not fall into the preset classes as "unknown" as soon as they are detected. According to the research [64], such a strategy is insufficient for real-world situations where robots frequently work in open spaces and deal with things they haven't been taught to. The importance of handling unknown instances has been recognized by earlier research, including Saxena et al.'s [54] work on grasping novel objects and cognitive studies on human vision. Although open-set identification has been the focus of certain CV techniques, these methods are frequently restricted to classification tasks or particular downstream robotics applications. The *Open-Set Instance Segmentation* (OSIS) network is designed for the identification of both known and unknown objects from point clouds. The paper details the problem formulation, where the goal is to map input features to a tuple representing instance and semantic labels, with a specific focus on handling unknown classes in the open-set setting. The proposed approach involves a CNN with shared backbone features, a detection head for known things, and an embedding head for instance-aware features. The inference procedure encompasses closed-set perception, associating points with prototypes, followed by open-set perception, classifying uncertain associations as unknown and clustering them into instances. The input representation involves a bird's eye view (BEV) rasterized image of a LiDAR point cloud, and the model incorporates a custom 2D convolutional feature pyramid network as its backbone. The detection head predicts anchor parameters, while the embedding head learns a category-agnostic embedding space. The closed-set perception utilizes prototypes for known things and stuff, with a subsequent step for identifying unknown instances through clustering.

In various perception systems like AD and robotics, the detection of 3D objects holds significant importance. The use of LIDAR as a sensor is prevalent for acquiring 3D point clouds in 3D object detection, mainly because of its resilience in diverse environmental conditions. That's the main purpose of the study [9], the open-set 3D object detection challenge occurs when models are applied to test datasets with object categories that were not seen during training. Unlike closed-set detection, the test dataset in open-set detection may include new or unexpected object categories. This task involves training a NN to recognize predefined classes while correctly identifying and

labeling objects from new, unseen categories. Challenges include potential misclassifications of known objects as unknown, the generation of false positive unknown objects, and the need for accurate bounding boxes for unfamiliar objects. The open-set 3D object detection challenge arises from the need to extend traditional 3D object detection models to accommodate scenarios where the test dataset may contain object categories not encountered during training. In classical closed-set 3D object detection, the class set of the test dataset aligns with that of the training dataset.

However, in real-world applications, such as AD, robotics, and surveillance, it is common for new or unexpected object categories to appear in the environment. The open-set 3D object detection problem involves training a NN not only to recognize predefined object classes but also to correctly identify and label objects from new and unseen categories. Handling unknown objects introduces complexities, including potential misclassifications of known objects as unknown, the generation of false positive unknown objects, and the challenge of producing accurate bounding boxes for objects not encountered during training. Addressing these issues is crucial for enhancing the robustness and practical applicability of 3D object detection systems in dynamic and diverse environments. To address issues with the naive open-set 3D object detector, the *MLUC* network, represented in 2.6 is presented. In this approach, the authors utilized the Euclidean distance sum (EDS) in metric space as a more reliable confidence score compared to the naive maximum softmax probability. By applying metric learning, there is high chance to identify more dependable regions containing unknown objects. Unsupervised clustering is then employed on each proposal region of unknown objects to refine their bounding boxes. Finally, Non-Maximum Suppression (NMS) is applied to filter out overlapping results, ensuring more accurate bounding boxes for unknown objects and reducing false positives.

2.2.4 Reconstructive approach

Masuda et al. [45] have introduced a method for discerning the anomaly status of an object point cloud. Unlike previous approaches, this technique focuses on the point clouds of individual enclosed objects. Given that automotive lidars capture comprehensive environmental scans, the challenge lies in initially extracting single objects or specific regions of interest through detection or clustering methodologies. The uniqueness of this approach lies in its consideration of isolated object point clouds, requiring a preliminary step to identify and extract these individual entities from the broader environmental data obtained by automotive LiDARs. The researchers developed a new *Variational Autoencoder* (VAE) model specifically for detecting anomalies in 3D point clouds. In the model's encoder, they included features like skip-connections and a graph max-pooling layer

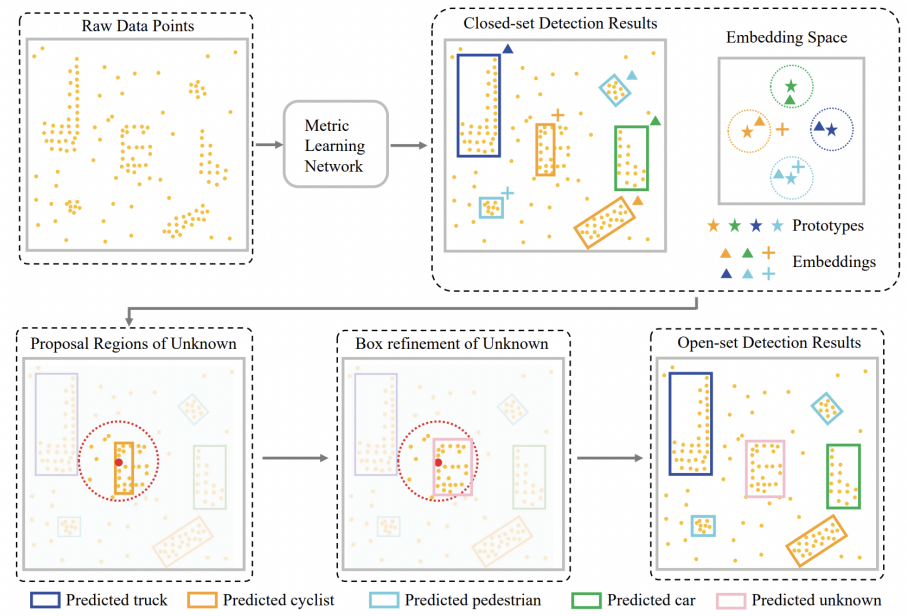

Figure 2.6: Pipeline of the MLUC network [9] for open-set 3D object detection. The architecture processes raw data points through a metric learning network to generate an embedding space, leading to closed-set detection results for known categories. Proposal regions for unknown objects are identified, refined, and ultimately included in the open-set detection results, allowing the system to detect both known and previously unseen object categories.

to better capture local details by using the structure of the data as a graph.

For the decoder, they were inspired by the FoldingNet [66] architecture but made a key change: instead of using a flat plane grid, they used a spherical grid shape for reconstruction. The encoder starts with an input matrix where each row represents a point's 3D position in space. Before the data is processed through the convolution layer, the model adds the local covariance matrix to each row.

The encoder outputs a matrix similar in size to the input, representing the reconstructed positions of the points. Importantly, the encoder also calculates the mean and variance for each point cloud, which the decoder then uses to reconstruct the point cloud by generating samples from the latent vector.

2.3 Mixed techniques studies

Multi-modal sensor signals offer a rich source of information for such tasks; however, integrating these high-dimensional and heterogeneous data streams poses significant challenges. Before introducing the main techniques of this section, it is of crucial importance to mention the work made by Caesar et al. [8] in the development of *nuTonomy scenes*, which we will refer to as nuScenes, a

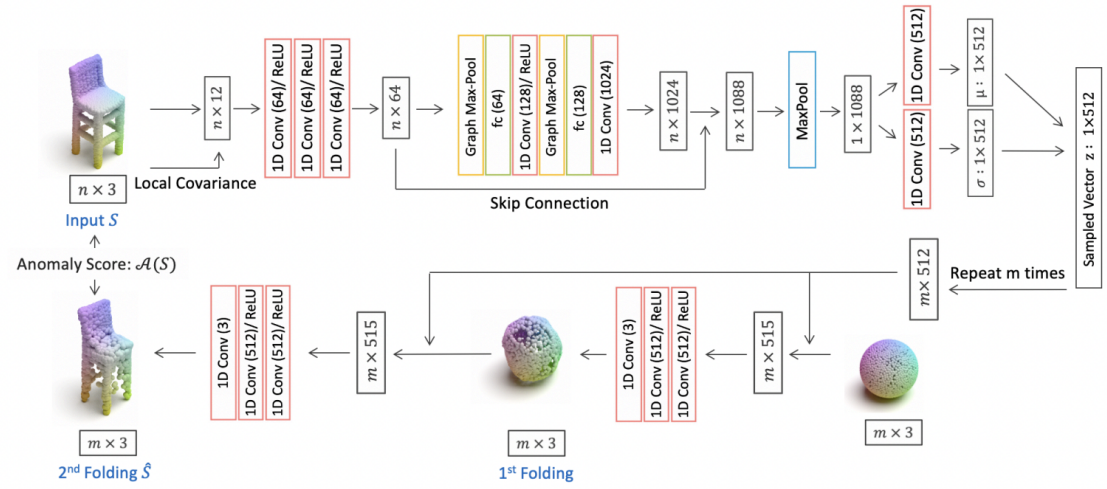

Figure 2.7: Representation of the model proposed by Masuda et al. [45]. Architecture is designed for anomaly detection in 3D point clouds, utilizing a VAE with a graph-based encoder and a folding-based decoder. The encoder incorporates local covariance, graph max-pooling, and skip connections to capture and compress the input data into a latent space. The decoder reconstructs the point cloud using a spherical grid and a two-step folding process. Anomaly detection is performed by calculating the anomaly score based on the difference between the input and reconstructed point clouds.

groundbreaking dataset designed specifically for autonomous vehicle research. nuScenes is the first dataset to include a complete autonomous vehicle sensor suite, featuring six cameras, five radars, and one lidar, all providing a full 360-degree field of view. The dataset contains 1000 scenes, each lasting 20 seconds, and is thoroughly annotated with 3D bounding boxes covering 23 different classes and 8 attributes. Compared to the pioneering KITTI dataset [21], nuScenes offers seven times more annotations and a hundred times more images. Due to these comprehensive features and the extensive annotations provided, nuScenes is quickly becoming the prime dataset in the field of multi-modal autonomous vehicle research.

First method suggested in [32] makes use of a *supervised variational autoencoder* (SVAE) model for failure identification in intricate, unstructured environments. In order to extract reliable features from high-dimensional inputs and combine the generative and discriminative models into a single training step, the SVAE makes use of the representational capability of VAEs. The suggested methodology uses a multi-class classifier to monitor the appearance of robot faults and identify possible causes in order to handle the anomaly and failure detection problem. The SVAE model is trained on data collected in field environments, where factors such as low localization accuracy between crop rows and uncertainties in sensory signals from LiDAR due to weeds and hanging leaves contribute to higher failure rates compared to controlled indoor settings. The anomaly detection system is constructed as a deep supervised multi-class classifier, which assumes the sensor data is

multi-modal, consisting of high-dimensional inputs (e.g., LiDAR) and low-dimensional inputs (e.g., wheel encoders). The AD module enables real-time failure identification and reaction by mapping the available sensor data to a corresponding class label at each time step. The two primary parts of the system's architecture are the *classifier* and the *feature generator* (FG) 2.8. In order to facilitate the classifier's training process, the FG greatly reduces the input dimensionality by mapping high-dimensional sensor inputs to latent variables. Through this procedure, the model is able to acquire reliable and significant representations of the initial inputs inside the latent space. These latent variables are then used by the classifier, which is usually a feed-forward NN, along with additional low-dimensional inputs, that can be derived modalities or raw sensor data, to accurately determine the kind and existence of anomalies. Tests conducted on actual field robot data show that the SVAE model generates comprehensible representations of the sensor data in addition to outperforming baseline techniques in terms of failure identification accuracy.

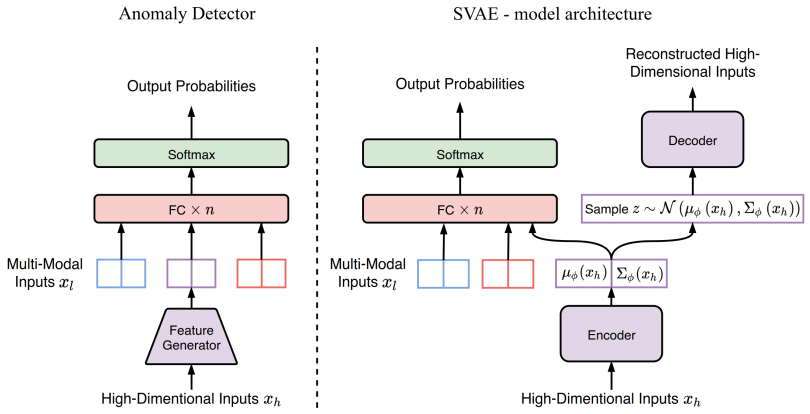

Figure 2.8: Model architecture proposed by Ji et al. [32]. The figure illustrates two components: The Anomaly Detector processes multi-modal inputs and high-dimensional inputs through a FG, followed by a fully connected layer, and produces output probabilities via a softmax layer. The SVAE model incorporates an encoder that generates a latent variable from high-dimensional inputs, which is then sampled to reconstruct the inputs via a decoder.

Chapter 3

Foundation

Sensors like cameras and LiDAR are the eyes of autonomous systems, enabling them to perceive and navigate their environments. However, even minor faults in these sensors can compromise system accuracy and safety. Traditional fault detection methods have limitations in handling dynamic faults.

3.1 Machine Learning Fundamentals

Understanding the methods employed in the process is made easier with the introduction of the ML baseline. *ML* represents a multifaceted discipline encompassing diverse learning paradigms that underpin its functioning. Among the most prominent are three foundational paradigms: *supervised learning*, *unsupervised learning*, and *reinforcement learning*, each contributing distinct methodologies to artificial intelligence's evolution. These paradigms serve as the bedrock of AI's learning methodologies, employed across various domains and applications.

In development, some techniques in ML models are core building blocks for the production of robust, reliable models, for example splitting into *training*, *validation*, *test* datasets. A training set is the dataset mainly used to train the model; the model learns through it the pattern and relationship of data and thus modifies its parameters to reduce errors. One, in certain ML paradigms, can think of a validation set as an independent dataset used during training to tune the model so that it does not overfit and is more likely to generalize well. The test set should be kept completely blind from both the training and the validation sets; it serves more like the final benchmark toward the model's performance and its generalization to new, unseen data.

The trade-off between *bias* and *variance* during model development is also crucial. While a model with high variance captures noise in addition to underlying patterns, it performs badly on

new data (overfitting), a model with strong bias oversimplifies the data, resulting in systematic errors (underfitting). Finding the ideal balance reduces both kinds of errors, ensuring the model generalizes well to new data. However, within the landscape of AI, the emergence of Artificial Neural Networks(NNs) and DL has significantly expanded the boundaries of these paradigms.

3.2 Neural Networks: Principles and Deep Learning

The notion of emulating the complex network of interconnected neurons found in the human brain has led to the development of a potent and adaptable paradigm known as Neural Networks (NNs). This introduction explores the architecture of NNs and delves into their basic principles, which form the foundation for a wide range of modern technologies. NNs have become indispensable, enabling a variety of tasks from speech and image recognition to sophisticated decision-making.

DL, a subset of Neural Networks, builds on these foundational concepts by leveraging multiple layers of neurons to automatically extract increasingly abstract features from raw data. This depth allows DL models to excel in handling large-scale, complex datasets, further advancing the capabilities of NNs.

3.2.1 Training Procedures in Neural Networks

Training in NNs is a complex and iterative process essential for achieving models that can make accurate predictions on diverse datasets. At its core, the training procedure involves fine-tuning the model's parameters to minimize a chosen loss function, which quantifies the dissimilarity between predicted outputs and true targets. Considering an initial dataset (X, Y), where X represents the input features, and Y denotes the corresponding target outputs.

The key in NN training lies in defining a score function or hypothesis function $f(X;\theta)$, where θ represents the model's parameters. This function transforms input features into predictions, forming the basis for model learning. The discrepancy between predicted values and actual targets is measured by the loss function, denoted as $L(Y, f(X;\theta))$. The selection of an appropriate loss function depends on the nature of the task, with Mean Squared Error (MSE) commonly used for regression and Cross-Entropy Loss for classification.

The optimization process aims to minimize this loss across the entire dataset. This is typically achieved using optimization algorithms like *Stochastic Gradient Descent* (SGD), where the model parameters θ are updated iteratively according to the gradient of the loss function with respect to the parameters. The update rule is represented by the formula:

$$\theta_{\text{new}} = \theta_{\text{old}} - \eta \nabla_\theta L(Y, f(X; \theta_{\text{old}}))$$

Here, η denotes the learning rate, controlling the step size in the parameter space, and ∇_θ represents the gradient. The learning rate is a critical hyperparameter, impacting the convergence and stability of the training process.

Training continues through multiple iterations (epochs), with the model gradually adapting its parameters to minimize the loss on the training data. Overfitting, where the model becomes too tailored to the training set, and vanishing/exploding gradients, which affect the stability of training in deep networks, are common challenges addressed during this process.

The validation set plays a crucial role by providing an independent dataset for assessing the model's generalization performance during training. Early stopping, regularization techniques, and the exploration of various architectures contribute to mitigating overfitting and improving model robustness.

3.2.2 Feed-Forward and Backpropagation

Effective learning in NNs is made possible by two basic mechanisms: **feedforward** and **backpropagation**. During feedforward, the data propagates through the network, get processed by every neuron, and then passed to the next layer until the output is generated. That is, computing the weighted summation of inputs with an activation applied afterward to introduce non-linearity.

On the other hand, backpropagation is an algorithm used to train the network by minimizing the disparity between the output predicted and the actual target. It enables the adjustment of model parameters by iteratively propagating errors backward through the network and updating weights to minimize the overall loss.

Consider a NN with L layers, each layer l having N_l neurons. Let $a_i^{(l)}$ represent the activation of the i-th neuron in layer l, and $z_i^{(l)}$ denote the weighted sum of inputs to that neuron. The forward pass computes activations $a_i^{(l)}$ using the input features X and the current weights $W^{(l)}$ and biases $b^{(l)}$ as follows:

$$z_i^{(l)} = \sum_{j=1}^{N_{l-1}} W_{ij}^{(l)} a_j^{(l-1)} + b_i^{(l)}, \quad a_i^{(l)} = \sigma(z_i^{(l)})$$

Here, $\sigma(\cdot)$ is the activation function.

The loss function L is computed based on the final layer's activations $a^{(L)}$ and the true targets Y. The goal of backpropagation is to minimize this loss. The gradient of the loss with respect to

the activations in the last layer is given by:

$$\delta_i^{(L)} = \frac{\partial L}{\partial a_i^{(L)}}$$

The backward pass then computes the gradient of the loss with respect to the weighted sum of inputs $z_i^{(l)}$ and the weights $W_{ij}^{(l)}$. The key recursive formula for the error term $\delta_i^{(l)}$ in each layer is:

$$\delta_i^{(l)} = \sigma'(z_i^{(l)}) \sum_{k=1}^{N_{l+1}} W_{ki}^{(l+1)} \delta_k^{(l+1)}$$

Here, $\sigma'(\cdot)$ is the derivative of the activation function.

The weight gradients are then calculated using the error terms:

$$\frac{\partial L}{\partial W_{ij}^{(l)}} = a_j^{(l-1)} \delta_i^{(l)}$$

And the biases are updated similarly:

$$\frac{\partial L}{\partial b_i^{(l)}} = \delta_i^{(l)}$$

Finally, the weights and biases are updated using an optimization algorithm like gradient descent:

$$W_{ij}^{(l)} \leftarrow W_{ij}^{(l)} - \eta \frac{\partial L}{\partial W_{ij}^{(l)}}, \quad b_i^{(l)} \leftarrow b_i^{(l)} - \eta \frac{\partial L}{\partial b_i^{(l)}}$$

Here, η is the learning rate.

It is of utmost importance to mention a subset of ML, DL, that has emerged as a transformative paradigm by enabling models to autonomously learn intricate representations from data hierarchies. In the field of sensor technology, DL represents a paradigm shift, especially when it comes to camera and LiDAR system problem detection. Its significance comes from its capacity to automatically deduce complex patterns and representations from unprocessed sensor data, enabling extremely sophisticated and effective problem detection systems. DL architectures like CNNs have demonstrated impressive performance in image analysis tasks, allowing cameras to recognize objects, complicated properties, and contextual information. Similar to this, DL models designed for spatial data analysis have made significant advances in the processing of LiDAR data, which frequently appears as point clouds. PointCloud-based networks, such as PointNet [51], have transformed the processing of LiDAR data by providing efficient detection of faults through strong spatial analysis. One indicator of DL models' applicability is how well they adjust to different

types of sensor data and failure patterns. Because they can extract complex representations from large amounts of data, they reduce the requirement for manually created features, which makes them extremely flexible and able to identify subtle errors that more conventional approaches could miss. Moreover, the ability to train consistently allows them to adjust to shifting problem scenarios, which improves the durability and adaptability of fault detection systems. DL requires a lot of labeled data, a lot of processing power, and performance optimization through fine-tuning. However, its significance lies not only in its present capacities but also in its propensity to develop further, providing prospective paths for improving sensor technology defect detection and boosting perception's resilience and dependability.

Convolutional Neural Networks (CNNs) stand as a cornerstone in vision-related tasks, fundamentally transforming how machines perceive and interpret visual data. CNNs leverage convolutional layers, which consist of learnable filters that slide across the input image to extract spatial hierarchies of features at various levels of abstraction. Early layers typically capture low-level features such as edges, textures, and simple shapes, while deeper layers progressively extract more complex and abstract features, such as parts of objects and, eventually, entire objects or patterns within the image.

This hierarchical feature extraction makes CNNs particularly well-suited for a wide range of computer vision tasks. In image classification, CNNs excel by learning to recognize and categorize objects or scenes within an image, assigning probabilistic labels to the input data. For object detection, CNNs not only classify objects but also predict bounding boxes, effectively localizing and identifying multiple objects within a scene. Semantic segmentation extends this capability by assigning a class label to each pixel in an image, thereby providing a detailed understanding of the spatial structure and composition of the scene. Pioneering architectures like LeNet, proposed by LeCun et al. [37], marked the inception of CNNs, introducing the concept of convolutional layers and pooling operations. Subsequent innovations, such as AlexNet's deeper architecture introduced by Krizhevsky et al. [36], revolutionized image classification tasks by exploiting deeper networks and rectified linear units (ReLU) for activation. VGGNet, with its emphasis on deeper architectures consisting of smaller filter sizes, further improved performance in image recognition tasks.

Residual Networks (ResNets) introduced by He et al. [25]. tackled the challenge of vanishing gradients in deeper networks by introducing skip connections, enabling training of extremely deep NNs. This innovation significantly boosted accuracy and led to the ability to train networks with hundreds of layers. The evolution of DL architectures extends beyond image classification. Architectures like *U-Net* [52] and its variations have proven highly effective in medical image segmentation, precisely delineating structures within medical scans.

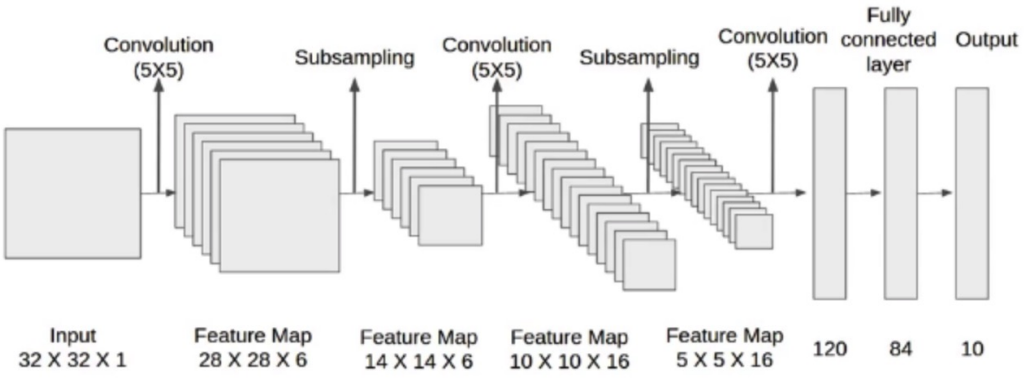

Figure 3.1: Architecture of the LeNet-5 model [37]. The network consists of two convolutional layers , each followed by a subsampling (pooling) layer, which progressively reduce the spatial dimensions of the feature maps while increasing the depth. After the convolutional and subsampling layers, the feature maps are fed into fully connected layers that ultimately produce the final output of class scores.

Additionally, *Mask R-CNN* introduced by He et al. [24]. combines object detection and instance segmentation, allowing for both locating objects and segmenting them at a pixel level, advancing the capabilities in CV tasks. The continual innovation and evolution of these architectures in DL have substantially elevated machines' abilities to understand and process visual information. Advancements in this domain have resulted in revolutionary discoveries across multiple domains, from enabling autonomous vehicles to interpret complex environments to aiding medical professionals in accurate diagnosis through enhanced medical imaging analysis.

3.2.3 Generative AI Techniques

Generative Artificial Intelligence stands at the forefront of innovation within the realm of DL, emphasizing the creation of new and original content autonomously. This branch of AI uses a wide range of models and methods to generate data in many domains, from images and text to music and even entire narratives.

Generative models like *GANs*, introduced by Goodfellow et al. [23], have reshaped the landscape of creative AI. GANs consist of two NNs, a generator and a discriminator, engaged in a competitive process. The generator aims to create data that is indistinguishable from authentic data, while the discriminator aims to differentiate between real and generated data. This adversarial training mechanism results in the generation of increasingly realistic and high-quality synthetic data. GANs have found applications in image synthesis, style transfer, and data augmentation, enabling the creation of hyper-realistic images and artistic transformations.

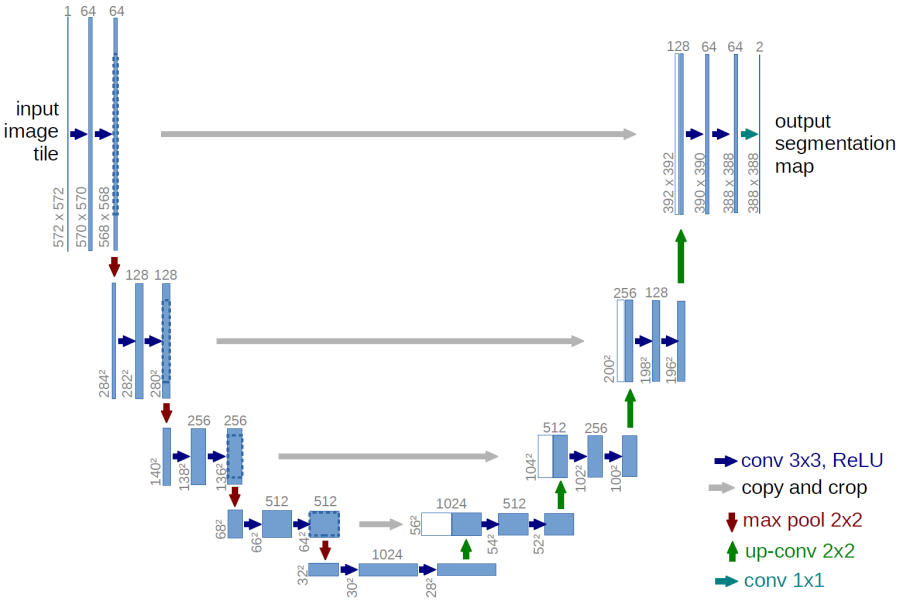

Figure 3.2: U-Net architecture [52]. The network follows a symmetric encoder-decoder structure, where the left side consists of a contracting path that captures context through a series of convolutional layers and max-pooling operations. The right side represents the expansive path, which recovers spatial resolution using up-convolutions and concatenation with corresponding feature maps from the contracting path, allowing precise localization. The final layer produces the output segmentation map.

VAEs, proposed by Kingma and Welling [35], are another class of generative models that learn latent representations of complex data distributions. VAEs work by encoding input data into a latent space and then reconstructing it back from this space, enabling the generation of new data samples. These models have been successful in diverse applications such as image generation, anomaly detection, and semi-supervised learning. Furthermore, autoregressive models like *Recurrent NNs* (RNNs) and *Transformers* have made substantial strides in language generation tasks. The Generative Pre-trained Transformer [60] (GPT) series, introduced by OpenAI, has demonstrated impressive capabilities in generating coherent and contextually relevant text, leading to advancements in natural language generation, dialogue systems, and content creation.

The impact of Generative AI extends beyond creative expression. It has implications in various fields, including drug discovery, where generative models aid in designing novel molecular structures, and in healthcare, assisting in generating synthetic medical data for training robust models without compromising patient privacy. The evolution and innovation in Generative AI continue to push the boundaries of what machines can create autonomously. However, ethical considerations surrounding generated content, such as misinformation and bias, remain crucial areas of research, emphasizing the need for responsible development and ethical frameworks in AI.

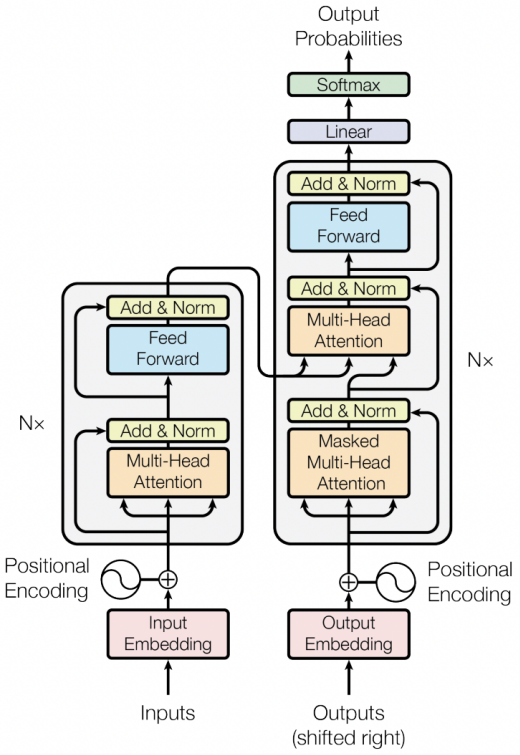

Figure 3.3: Transformer architecture proposed by Vaswani et al's. [60]. The model consists of an encoder-decoder structure, where both the encoder and decoder are composed of multiple layers of Multi-Head Attention and Feed-Forward Neural Networks, each followed by Add & Norm operations. The encoder processes the input embeddings, enhanced with positional encoding, through several layers of self-attention mechanisms and feed-forward networks. The decoder, similarly structured, also incorporates masked multi-head attention to process the output embeddings, which are shifted right.

3.3 Camera Technology Overview

Camera sensor technology forms the backbone of modern digital imaging systems, converting light into electronic signals that can be processed to produce images. This technology has evolved significantly over the years, driven by advancements in semiconductor materials, fabrication techniques, and the demand for higher image quality and smaller, more efficient devices.

3.3.1 Types of Image Sensors

There are primarily two types of image sensors used in cameras: *Charge-Coupled Devices (CCDs)* and *Complementary Metal-Oxide-Semiconductors (CMOS)*. Both of these sensors perform the same basic function of capturing light and converting it into an electrical signal but they do so using different technologies and with different implications for performance and application.

Charge-Coupled Devices CCDs were the first to be widely used in digital cameras. In a CCD sensor, the charge generated by incident light in the photodiodes is transferred through the chip to a corner of the sensor, where it is converted into a voltage, digitized, and stored. CCDs are known for their high-quality image output with low noise, which is why they were favored in early digital cameras and applications requiring high image fidelity, such as astronomy and medical imaging.

However, CCDs are relatively power-hungry and expensive to manufacture due to the complex process of moving charges through the sensor. Furthermore, they have limited readout speeds, making them less suitable for applications requiring rapid image capture, such as video.

Complementary Metal-Oxide-Semiconductors CMOS sensors [20], on the other hand, have become the dominant technology in modern digital cameras due to their lower power consumption, faster readout speeds, and greater integration capabilities. Unlike CCDs, where the charge is moved across the chip, CMOS sensors allow each pixel to be read individually. This architecture enables faster data readout, which is particularly beneficial for high-speed photography and video.

CMOS sensors are also less expensive to produce because they can be fabricated using standard semiconductor manufacturing processes, allowing for the integration of additional circuitry on the same chip, such as amplifiers, noise-correction, and analog-to-digital converters (ADCs).

However, CMOS sensors traditionally suffered from higher noise levels compared to CCDs. Recent advancements, such as Back-Side Illumination (BSI) technology and improved noise reduction algorithms, have mitigated these issues, making CMOS sensors competitive in terms of image quality.

3.3.2 Sensor Characteristics

The performance of a camera sensor is determined by a combination of several key characteristics. These characteristics define how effectively the sensor can capture and reproduce images under various conditions.

Pixel Size and Density The size of each pixel on the sensor affects both the resolution of the image and the sensor's light-gathering capability. Smaller pixels allow for higher resolution images but may result in lower sensitivity to light and increased noise, particularly in low-light conditions. Conversely, larger pixels can capture more light, improving performance in low-light environments but at the cost of reduced resolution.

Dynamic Range Dynamic range is the sensor's ability to capture details in the darkest and brightest parts of the image simultaneously. Sensors with a higher dynamic range can produce images with better detail in both shadows and highlights. This characteristic is critical in scenes with significant contrast, such as landscapes with bright skies and dark shadows.

Quantum Efficiency (QE) Quantum efficiency measures the percentage of photons hitting the sensor that are converted into electrons. A higher QE indicates a more efficient sensor, capable of capturing more light and producing images with less noise. Advances in sensor technology have steadily improved QE, particularly with the introduction of BSI sensors, which reduce the loss of light that occurs in front-side illuminated sensors.

Noise Performance Noise in camera sensors arises from several sources, including shot noise (due to the statistical nature of photon capture), thermal noise, and read noise. Reducing noise is critical for producing high-quality images, particularly in low-light conditions. CMOS sensors have seen significant improvements in noise reduction, including the implementation of correlated double sampling (CDS) to reduce read noise and the use of better materials and fabrication techniques to minimize thermal noise.

3.3.3 Camera Data Collections

The realm of perception in highly automated driving is enriched by a multitude of datasets catering to various facets of camera-based analysis. Notably, datasets like the Cityscapes dataset [13], BDD100k dataset [67], openDD dataset [6], and Mapillary Vistas dataset [46] stand as pillars, offering vast collections of camera images or sequences essential for visual perception tasks. These

datasets are instrumental in providing ample annotations conducive to tasks like semantic and instance segmentation, as well as object detection, fostering the development of robust algorithms for autonomous systems. Moreover, the landscape expands to encompass datasets explicitly designed for the nuanced task of unknown object detection from images, such as the RoadAnomaly [40], Lost&Found [49], and Fishyscapes [3] datasets. These specialized repositories serve as invaluable resources for training models to identify and handle anomalies or unfamiliar objects within visual data, crucial for enhancing the adaptability of autonomous systems. Erdoğmuş et al. [19] proposed a dataset created injecting faults to the camera sensors of the ROKOS robot control system including various techniques such as dilation, which enlarges highlighted regions, and erosion, which reduces them. Then, the open method combines erosion followed by dilation to remove small objects while preserving larger structures, whereas the close method involves dilation followed by erosion to close small gaps and smoothen boundaries. The gradient method, which calculates the difference between dilation and erosion, is used to highlight object boundaries. Additionally, motion-blur introduces a blur effect to simulate motion, and partial loss methods selectively destroy portions of an image, such as horizontal, vertical, or patterned losses.

3.3.4 Projective geometry

Projective geometry is fundamental in the realms of camera calibration and the fusion of LiDAR and camera data. At its foundation, it offers a mathematical framework that helps comprehend the viewpoint and spatial relationships present in imaging systems. In the context of camera calibration, projective geometry is instrumental in mapping the three-dimensional world onto a two-dimensional image plane accurately. This calibration is crucial for tasks such as accurate distance measurement, object recognition, and spatial reconstruction. Additionally, when integrating LiDAR and camera data, projective geometry serves as a key foundation for aligning and synchronizing information from these disparate sensors.

2D Affine Transforms Affine transforms are fundamental geometric operations that preserve points, straight lines, and parallelism. In 2D space, an affine transformation can be represented using the following matrix equation:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Here, (x, y) are the coordinates of the original point, (x', y') are the coordinates of the transformed point, and the 2×3 matrix represents the affine transformation matrix with parameters a, b, c, d, e, f. The last row in the matrix is always $[0, 0, 1]$ to maintain the homogeneous coordinates. The transformation matrix can be decomposed into different operations, combining these operations allows for complex transformations:

$$\text{Translation: } \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Rotation: } \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{Scaling: } \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2D Projective Transforms (Homography). Projective transforms, or homographies, are more general transformations that include perspective distortions. In 2D, a projective transformation can be represented as:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Here, (x, y) are the coordinates of the original point, (x', y') are the coordinates of the transformed point, and w' is a homogeneous coordinate. The 3×3 matrix represents the homography matrix with parameters h_{ij}. The projective transform can represent operations such as rotation, scaling, translation, and perspective. A common use is in image stitching, where homographies are used to align and merge multiple images.

Relationship between Affine and Projective Transforms.

An affine transform is a subset of a projective transform. If the last row of the projective transformation matrix is $[0, 0, 1]$, it reduces to an affine transformation. Affine transformations are

essentially a special case of projective transformations where parallel lines remain parallel and ratios of distances along parallel lines are preserved. In summary, 2D affine and projective transforms are powerful tools for geometric manipulation of images or objects. They are commonly used in CV, computer graphics, and image processing to achieve tasks like image registration, correction, and stitching. Understanding the underlying matrix representations and their applications is essential for working with these transformations in various fields.

Homogeneous Coordinates and Projective Space. In computer graphics, CV, and geometric transformations, homogeneous coordinates and projective space are essential concepts. Homogeneous coordinates extend the traditional Cartesian coordinates to include an additional dimension, enabling the representation of points at infinity and facilitating projective transformations.

Homogeneous Coordinates.

Traditional Cartesian coordinates are represented as (x, y), but homogeneous coordinates introduce an extra coordinate, often denoted as w. The 2D homogeneous coordinates are represented as (x, y, w), and the 3D homogeneous coordinates as (x, y, z, w). The key property of homogeneous coordinates is that a point represented as (x, y, w) is equivalent to $(\lambda x, \lambda y, \lambda w)$ for any non-zero scalar λ. Homogeneous coordinates are useful for representing points at infinity, as a point at infinity can be represented with $w = 0$.

Homogeneous Matrix Representation. Homogeneous transformations are often represented using matrices. For a 2D transformation, the homogeneous matrix looks like:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ p & q & r \end{bmatrix}$$

Here, the matrix represents both translation (by (p, q)) and linear transformations (by (a, b, c, d, e, f)). For 3D transformations, the matrix becomes a 4x4 matrix.

Projective Space It is an extension of Euclidean space that includes points at infinity. Projective transformations are used to model perspective transformations, such as those seen in cameras. Projective space is particularly crucial when dealing with the projection of 3D points onto a 2D plane, as it allows for a unified representation of both finite and infinite points.

Homogeneous Coordinates and Projective Transformations Projective transformations, can be represented using homogeneous coordinates. The transformation of a point (x, y, w) in projective space can be expressed as:

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Here, h_{ij} are the elements of the homogeneous transformation matrix.

Camera calibration Matrix The camera calibration matrix is a fundamental component, crucial for accurately mapping 3D points in the world onto a 2D image plane. It forms the bridge between the world coordinates of a scene and the pixel coordinates in an image. The central concept in camera calibration is central projection, which assumes a pinhole camera model. In this model, light rays from a 3D point in the world pass through a single point, the optical center or camera center, and intersect with the image plane. The camera calibration matrix is often denoted as K and is given by:

$$K = \begin{bmatrix} f_x & s & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Here, f_x and f_y represent the focal lengths of the camera in the x and y directions, (c_x, c_y) is the principal point (the point where the optical axis intersects the image plane), and s denotes skew.

Anisotropic Scaling and Skew Anisotropic scaling refers to the situation where the focal lengths in the x and y directions are different ($f_x \neq f_y$). This can occur due to manufacturing imperfections or deliberate design choices. Anisotropic scaling is accommodated in the camera calibration matrix, allowing for precise correction and compensation in the transformation from 3D to 2D space.

Skew (s) in the camera calibration matrix represents the non-orthogonality between the image axes. In well-calibrated cameras, skew is usually zero, indicating that the image axes are perpendicular. However, in certain situations, such as lens misalignments, skew may be non-zero. Accurate calibration accounts for skew in the camera matrix to ensure geometric accuracy in subsequent CV tasks.

Incorporating Distortion Real-world cameras introduce radial and tangential distortion, further complicating the camera model. Radial distortion causes straight lines to appear curved, especially towards the edges of the image. Tangential distortion occurs when the lens is not perfectly parallel to the image plane, causing warping effects. Both distortions can be modeled and corrected through additional terms in the camera calibration matrix.

$$K' = \begin{bmatrix} f'_x & s' & c'_x & 0 \\ 0 & f'_y & c'_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Here, f'_x and f'_y are the distorted focal lengths, (c'_x, c'_y) is the distorted principal point, and s' represents distorted skew. Understanding the camera calibration matrix, central projection, anisotropic scaling, skew, and distortion parameters is essential for accurate geometric transformations in CV applications. Calibration provides the means to rectify images, correct distortions, and accurately

map the 3D world onto a 2D image plane, forming the foundation for various CV tasks, including object recognition, 3D reconstruction, and augmented reality.

Camera calibration involves determining the internal (intrinsic) and external (extrinsic) parameters that characterize the relationship between the 3D world and the 2D image captured by the camera. This process is crucial for CV tasks, such as object recognition, 3D reconstruction, and augmented reality.

3.3.5 Intrinsic Parameters

Intrinsic parameters are internal characteristics of the camera that remain constant regardless of the camera's position or orientation in the scene. The intrinsic parameters are usually organized into the camera calibration matrix K. They define the internal geometry and optical properties of the camera. The primary intrinsic parameters include:

- **Focal Length (f_x, f_y):** Represents the distance from the camera's optical center to the image plane, determining the scale of the image.
- **Principal Point (c_x, c_y):** Specifies the location of the image center, the point where the optical axis intersects the image plane.
- **Skew (s):** Describes any non-orthogonality between the image axes. For well-calibrated cameras, skew is usually zero.
- **Distortion Parameters:** Model radial and tangential distortions introduced by the lens. Commonly represented as k_1, k_2, p_1, p_2, k_3.

3.3.6 Extrinsic Parameters

Extrinsic parameters define the camera's position and orientation in the 3D world. They are essential for mapping 3D points in the world to their corresponding 2D image coordinates. The primary extrinsic parameters include:

- **Rotation Matrix (R):** Describes the rotation of the camera in the world coordinate system. For a 2D rotation, the matrix can be expressed as:

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

where θ is the angle of rotation. This matrix rotates a point (x, y) around the origin to a new point (x', y') given by the matrix multiplication:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = R(\theta) \begin{pmatrix} x \\ y \end{pmatrix}$$

- **Translation Vector (T):** Specifies the position of the camera's optical center in the world coordinate system.

The extrinsic parameters are often combined into a 3x4 matrix called the extrinsic matrix $[R|T]$.

$$P_{\text{world}} = K \cdot [R|T] \cdot P_{\text{object}}$$

Here, P_{world} is the 2D image point in pixel coordinates, P_{object} is the corresponding 3D point in world coordinates.

3.4 LiDAR Technology Overview

LiDAR, through its emission of laser pulses and subsequent measurement of their reflections, generates detailed and precise 3D point clouds. These point clouds furnish an accurate representation of the environment, allowing for real-time distance estimation, object localization, and environmental mapping. LiDAR's ability to create a comprehensive spatial model aids in navigating complex and dynamic surroundings, contributing significantly to applications like autonomous vehicles, robotics, and geographical mapping.

3.4.1 Principles of operation

LiDAR technology is a sophisticated remote sensing method that relies on the emission of laser pulses, meticulously engineered to measure precise distances, object geometries, and intrinsic surface characteristics within a broad field of view. The operational intricacies of LiDAR extend beyond the basic emission of laser pulses, encompassing critical elements such as wavelength selection—a pivotal factor that directly influences the system's accuracy, range, and overall performance depending on the specific application requirements.

Choosing the correct wavelength isn't a simple decision. It requires careful consideration of how light interacts with the environment, including factors like how the atmosphere absorbs and

scatters light, and how different surfaces reflect it. This choice is crucial for ensuring that the LiDAR system can accurately capture detailed information about the objects it scans.

The detection of reflected laser pulses is another essential aspect of LiDAR. The system uses detectors, such as avalanche photodiodes (APDs) or photomultiplier tubes (PMTs), to convert the light that bounces back into electrical signals. These detectors are particularly important because they need to be sensitive enough to detect even weak signals, especially over long distances.

A core principle of LiDAR is the time-of-flight (ToF) measurement, which calculates the distance to an object by measuring the time it takes for a laser pulse to travel to the object and back. Accurate ToF measurements are vital for determining distances and rely on precise timing and signal processing.

Calibration is also a crucial process in LiDAR systems. It helps to reduce errors that can arise from sensor noise, interference, or slight misalignments in the system. By calibrating the system, adjusting components like the laser, detectors, and timing mechanisms, LiDAR ensures that its measurements remain consistent and accurate over time.

3.4.2 Intrinsic Parameters

In this chapter, we explore the critical intrinsic parameters that define the performance and capabilities of LiDAR (Light Detection and Ranging) sensors. These parameters play a vital role in determining how effectively a LiDAR system can capture and interpret the surrounding environment.

- **Range Resolution.** Range resolution is a critical intrinsic parameter determining the ability of a LiDAR sensor to distinguish between objects at different distances. It is expressed in meters and depends on the pulse width of the laser. Smaller pulse widths enable finer discrimination between closely spaced objects along the line of sight.

- **Field of View (FOV).** FOV defines the angular extent of the environment that a LiDAR sensor can observe, typically measured in degrees. A wider FOV allows the sensor to capture a larger portion of the scene in a single measurement, aiding in applications like autonomous vehicles where a comprehensive understanding of the surroundings is crucial.

- **Angular Resolution.** Angular resolution is the ability of the LiDAR sensor to distinguish between different angles or directions. It's expressed in degrees and influences the density and precision of the acquired point cloud. Higher angular resolution enables the sensor to capture more detailed information about the surroundings.

- **Wavelength.** The wavelength of the laser used by the LiDAR system affects its performance, especially in challenging environmental conditions. Different wavelengths interact differently with atmospheric particles, affecting the system's range and accuracy. Shorter wavelengths, such as in the UV spectrum, are often used to enhance performance.

- **Pulse Repetition Rate (PRR) / Pulse Rate.** PRR denotes the number of laser pulses emitted per second. It directly impacts the point cloud density and the speed at which the LiDAR sensor can acquire data. Higher PRR values are beneficial for capturing detailed information at higher speeds.

3.4.3 Extrinsic Parameters

In this section, we explore key external parameters that influence the performance and accuracy of LiDAR systems in real-world applications. Understanding these factors is crucial for ensuring that LiDAR data is accurately captured, interpreted, and utilized across various domains.

- **Sensor Position.** The 3D coordinates (x, y, z) of the LiDAR sensor in the external coordinate system. Precise knowledge of the sensor's position is crucial for accurately georeferencing the acquired point cloud data.

- **Sensor Orientation.** The angles representing the LiDAR sensor's orientation in the external coordinate system. These angles, often expressed as pitch, yaw, and roll, determine the sensor's orientation relative to the external world.

- **Time Synchronization.** Ensuring that LiDAR data is synchronized with other sensors (such as cameras or IMUs) is crucial for accurate sensor fusion. Time synchronization allows for the correlation of data from different sensors, enabling a comprehensive understanding of the environment and improving the overall accuracy of the system.

- **Accuracy.** Accuracy is the precision with which a LiDAR sensor measures distances. It represents how closely the measured values align with the true values. Accurate LiDAR data is essential for applications where accurate distance measurements are critical, such as in mapping or surveying.

- **Precision.** Precision refers to the repeatability of measurements taken by the LiDAR sensor. It indicates how consistently the sensor can reproduce the same measurement under identical conditions. High precision is crucial for reliable and consistent performance in various applications.

3.4.4 LiDAR Data Collections

In the lasts years multi-modal datasets has expanded significantly, complementing the traditional camera-based datasets. Among these, the KITTI [21] dataset stands out for its provision of both camera and LiDAR data. However, this fusion of sensory inputs extends beyond KITTI, evident in a diverse array of datasets such as Apolloscape [28], A2D2 [22], Waymo Open [56], PandaSet [65], KAIST [31] and ShapeNET [11]. These datasets showcase the synergy between LiDAR and camera data, offering invaluable insights into various scenarios. Notably, some datasets, including Apolloscape [28] and A2D2 [22], go beyond 3D bounding boxes, delving into point cloud semantic segmentation, enriching the depth of understanding. Moreover, the Canadian Adverse Driving Conditions [50] dataset is a unique addition, specifically focusing on image and LiDAR data captured amidst wintry weather conditions, addressing a niche yet crucial aspect of real-world environments. This diversification underscores the burgeoning complexity and richness of multi-modal datasets in advancing research and applications in CV and autonomous systems.

3.5 Sensor Fusion Techniques

Many approaches are used in the complex topic of sensor fusion to optimize the relationships between camera and LiDAR data, overcoming obstacles and improving applications. Using the color and texture features from the camera along with the geometry insights from LiDAR, feature-level fusion combines the information obtained from both sources. This combination strengthens item detection and recognition, resulting in a deeper understanding of the scene. Decision-level fusion reduces errors from individual sensors and increases overall system reliability by combining decisions from multiple sensors into a final decision. In order to create a fused representation in raw data, data-level fusion combines the LiDAR point cloud with the camera image. Taking into account the complexities of sensor fusion, temporal misalignment occurs when cameras and LiDAR sensors function at different frequencies, requiring synchronization methods like interpolation and timestamping to align their temporal data. Accurate calibration between the camera and LiDAR coordinate systems is required in order to address spatial misalignment simultaneously. In order to guarantee accurate spatial coordination, complex calibration procedures that include both extrinsic and internal calibration are used. This is because alignment errors might add inaccuracies during fusion.

3.5.1 3D Points to 2D Image Plane Projection

A sequence of mathematical procedures is used for this purpose. The processes in this procedure are as follows: first, projecting the 3D points onto the 2D image plane; second, converting the points from the world coordinate system to the camera coordinate system.

Transforming 3D Points to Camera Coordinates First, we need to transform the 3D points from the world coordinate system to the camera coordinate system. This transformation involves a rotation and a translation, which can be represented by a 3x4 extrinsic matrix $[R|t]$:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

where:

- (X_w, Y_w, Z_w) are the coordinates of the 3D point in the world coordinate system.
- (X_c, Y_c, Z_c) are the coordinates of the 3D point in the camera coordinate system.
- R is a 3x3 rotation matrix.
- t is a 3x1 translation vector.

Projecting onto the 2D Image Plane Next, we project the 3D points in the camera coordinate system onto the 2D image plane. This step uses the intrinsic camera matrix K:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix}$$

where K is the 3x3 intrinsic matrix:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

and:

- f_x and f_y are the focal lengths of the camera in the x and y directions, respectively.
- c_x and c_y are the coordinates of the principal point (the optical center) in the image plane.

Combining the intrinsic and extrinsic matrices, the overall projection matrix P can be expressed as:

$$P = K[R|t]$$

Final Projection The final projection of a 3D point (X_w, Y_w, Z_w) from the world coordinate system to the 2D image plane (u, v) is given by:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

This can be written as:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

After multiplying these matrices, we get the 2D coordinates (u, v):

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x(r_{11}X_w + r_{12}Y_w + r_{13}Z_w + t_x) + c_x(r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z) \\ f_y(r_{21}X_w + r_{22}Y_w + r_{23}Z_w + t_y) + c_y(r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z) \\ r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z \end{bmatrix}$$

To obtain the final 2D pixel coordinates (u, v), we normalize by the third component:

$$u = \frac{f_x(r_{11}X_w + r_{12}Y_w + r_{13}Z_w + t_x) + c_x(r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z)}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z}$$

$$v = \frac{f_y(r_{21}X_w + r_{22}Y_w + r_{23}Z_w + t_y) + c_y(r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z)}{r_{31}X_w + r_{32}Y_w + r_{33}Z_w + t_z}$$

These 2D coordinates were then saved into files, whereby a detailed numeric expression of point cloud data in 2D is carefully saved. The 2D coordinates were then used in image projection 4.6. This step involves visualizing each point of the 3D onto the image plane in its 2D form.

3.5.2 Deep Neural Networks in Sensor Fusion

Deep NNs, also, can be made to directly receive data from both sensors and can be trained to understand intricate relationships for a variety of applications. This end-to-end method has demonstrated promising performance in semantic segmentation and object detection tasks. Deep networks are capable of fusion at various levels. At the input layer, early fusion combines raw sensor data, whereas late fusion combines characteristics from higher layers.

Chapter 4

Anomaly Detection Datasets Analysis

In our pursuit of enhancing AD capabilities, we have constructed a comprehensive database using data taken from onboard sensors and others open-source datasets. Leveraging the onboard camera's vantage point, we are curating a diverse and expansive repository of real-world driving scenarios, encompassing a spectrum of environments, lighting conditions and others situations that may arise. This database is meticulously designed to capture various corner cases, adverse weather conditions, occlusions, and unexpected scenarios that autonomous vehicles might encounter on the roads.

4.1 Building Our Dataset

The creation of this dataset is based on the concept of injecting faults that can be found in the ordinary workflow of camera and LiDAR sensors. As stated in the previous paragraph, the introduction of a fault in camera sensors can be made through various techniques like the most common image processing or developing a model of generative artificial intelligence to get a more realistic result. Basically, the image processing way is easier to implement and can produce good output, in order to get useful images we have to deal with different techniques based on the type of fault we want to introduce. Image processing techniques are crucial, encompassing methods like image filtering, pixel manipulation, and geometric transformations to induce distortions. Beyond single-fault injections, researchers may explore the combination of multiple faults to mimic complex scenarios.

4.1.1 Camera Data

In order to generate an high quality and well balanced dataset for our task three different sources have been selected, each contributing varying numbers of images. The total number of images in the dataset is *55,890*, distributed as follows:

KITTI Dataset

- **Number of images:** 47,885
- **Description:** KITTI is a popular benchmark dataset for joint research into AD. It archives a vivid collection of urban scenes taken at different days and times of the day, hence in various weather conditions.

Canadian Adverse Conditions Dataset

- **Number of images:** 7,000
- **Description:** The Canadian Adverse Conditions dataset includes images that were taken during such weather conditions as snow, rain, and fog.

Our Data Sources

- **Image Count:** 1,005
- **Description:** This subset of images is collected from our data acquisition efforts. These images are captured about situations relevant for meeting our research needs.

The curated dataset is so composed of two exact halves, one full half of the unaltered images was to represent typical driving conditions images, from now we refer to them as *original images*. The second half of the dataset was prepared for a fault injection process simulating a series of various real-world visual anomalies that could be met by an AD system, these images will be referred as *faulty images*. For the second half, between one and four types of faults were selected and injected randomly on a per-image basis. This lead to a total dataset size of *111780* images.

Types of Fault 11 Types of fault have been injected into the original images. The type and number of injected faults were randomly selected to avoid noticeable patterns. Among the injected faults are:

- **Gaussian noise**, replicating sensor noise.

- **Hot pixels**, mimicking sensor errors.
- **Gaussian blur**, simulating out-of-focus images.
- **Random rotation**, to test the system's response to changes in orientation.
- **Color aberrations**, distorting colors.
- **Occlusions**, Random blackout of rectangular areas.
- **Jitter effects**, achieved by randomly displacing pixels.
- **Lens flares**, simulating bright light artifacts.
- **Change of exposure**, introducing different lighting conditions.
- **Barrel and pincushion distortions**, testing the system's ability to cope with lens distortion effects.

For each of the enumerations, a well-designed and implemented type has been designed to make results as close to reality as possible and offer a challenging test environment for the algorithms of AD. In this way, we achieved a balanced distribution of anomalies in the fault-injected half of the dataset. Each image with faults is well-represented, creating a nearly even distribution. This ensures the dataset is comprehensive, covering a wide range of visual impairments, making it a valuable resource for developing, testing, and validating AD systems in various real-world conditions.

The distribution of images with only one fault injected is presented in fig. 4.1. This distribution indicates a relatively balanced spread of each fault type, with Gaussian blur being the most common and color aberrations the least. In addition to single-fault images, in fig. 4.2 is presented an histogram that describe the distribution of number of faults injected into the original images. This complementarity in the type of fault combinations would in general improve the robustness of the dataset while validation would be through fault-detection and correction algorithms. An overview figure of the faults distribution is shown in figures 4.3 and 4.4. The number of images injected with one specific fault fluctuate near the figures of 12500–12900, whereby no fault type overweighs or under-represents another, hence making the dataset evidently well rounded and fit for analysis. In general, therefore, this dataset presents a rich variety of faults on their own or in combination, making it very resourceful in the development and testing of image processing algorithms for fault detection, correction, and enhancement. The balance present in the representation of each type of fault and the substantial number of images with several faults suggest comprehensive testing scenarios that may be well close to the real-world situation.

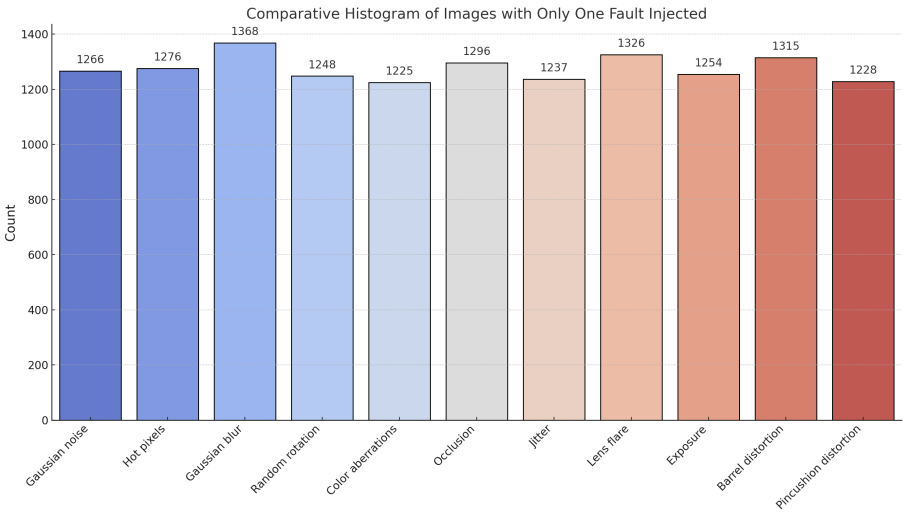

Figure 4.1: Distribution of Images with only one fault injected

4.1.2 LiDAR data

In our dataset, LiDAR point clouds are integral to producing accurate 2D projections, which are essential for analyzing and interpreting spatial data. We deliberately chose to keep the LiDAR data unaltered to maintain its precision and reliability, which are crucial for generating 2D images that accurately represent the 3D structures captured by the LiDAR sensors. This decision ensures that the spatial relationships and details observed in the real world are faithfully replicated in the 2D projections, making the LiDAR data a dependable reference point throughout the dataset.

The enrichment of our dataset involved a meticulous process of projecting the 3D LiDAR point clouds into the 2D camera space. This projection yielded two primary outputs: first, projective image versions that provide a visual representation of the spatial data, where each pixel corresponds to a specific point in the 3D environment; second, detailed 2D coordinate files that contain the exact coordinates of these projections, offering highly precise numerical data.

To ensure comprehensive coverage, we included LiDAR data for every camera image in the dataset. This means that for each camera image, there is a corresponding projective image generated from the LiDAR point cloud, along with a 2D coordinate file. This pairing allows for a seamless integration of visual and spatial data, enhancing the dataset's utility across various applications.

When it comes to fault-injected camera images, the corresponding LiDAR data remains unaltered. This approach serves a dual purpose: it preserves the LiDAR data as a reference point, enabling a clear comparison between the accurate spatial information captured by the LiDAR and the potentially distorted visual data. By comparing fault-injected images with their corresponding unaltered LiDAR projections, researchers can better understand the impact of visual impairments

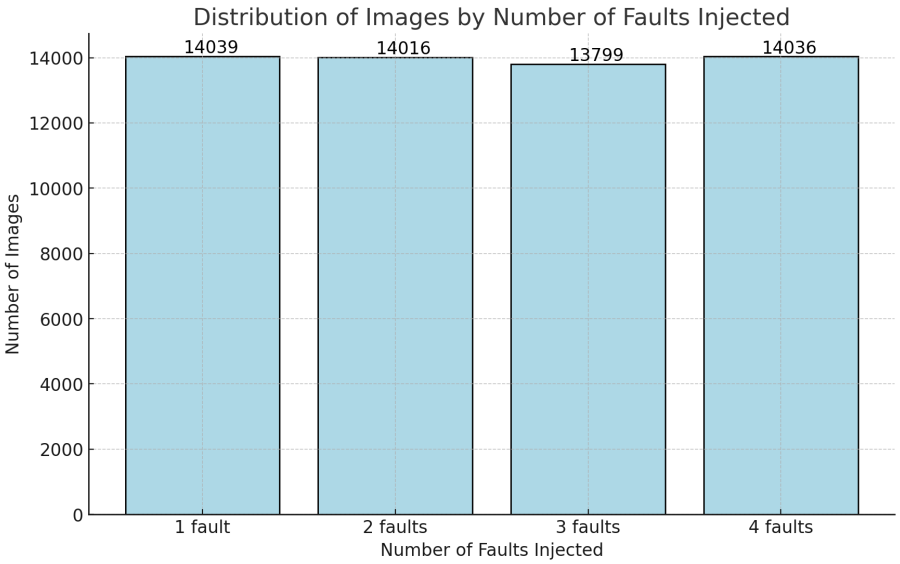

Figure 4.2: Distribution of Number of Faults Injected Into our Own Dataset

on spatial perception and assess how these faults might affect the performance of systems that rely on visual data. The unaltered LiDAR data provides a reliable baseline against which the accuracy and robustness of the systems can be evaluated.

Furthermore, example images from the dataset illustrate the relationship between the camera images and the LiDAR data are given in fig. 4.5. This example include a standard 2D camera image and its corresponding projective image from the LiDAR point cloud.

Moreover, the view from where the data was captured varied across the sensors. It is the case that variables of the setup including height, angle, and position of the LiDAR sensors in each dataset resulted in a variation of view from which the areas were captured and covered. Since all of these source datasets follow differing conventions, the first step of real importance in using LiDAR data from these sources was its standardization to uniformity and compatibility.

The next step after the normalization of the LiDAR data was the conversion of 3D point clouds into 2D coordinate spaces. The detail about the implementation can be found in 3.5.1 and in figure 4.6 is shown one example of projection.

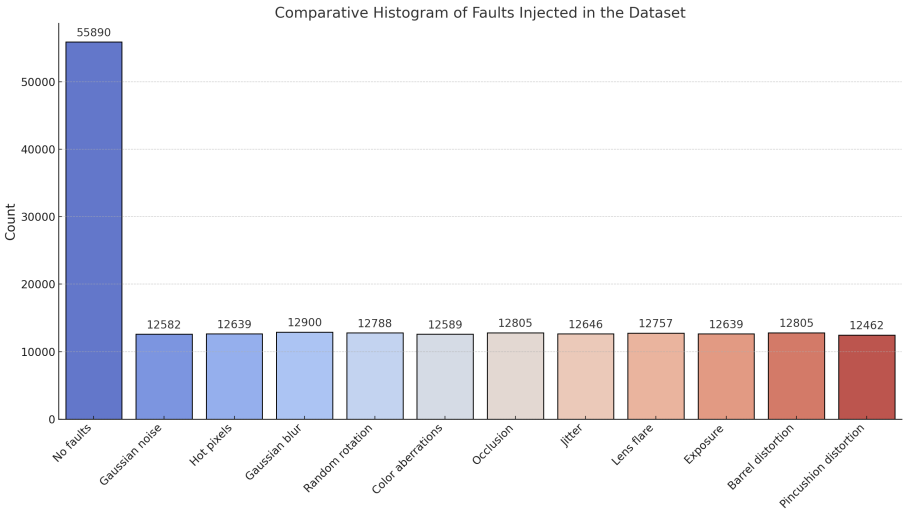

Figure 4.3: Distribution of Various Fault Classes Injected Into our Own Dataset

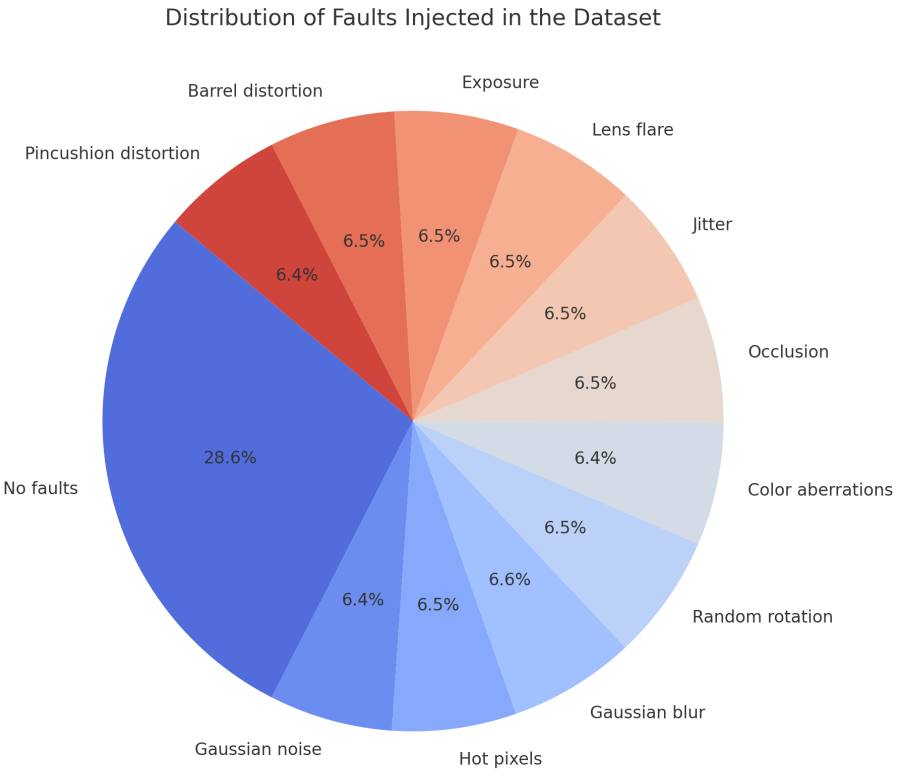

Figure 4.4: Pie Chart of Distribution of Various Fault Classes Into our Own Dataset

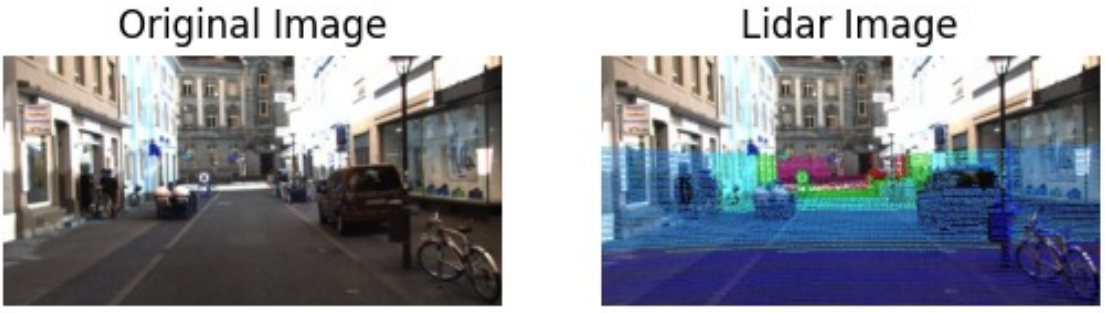

Figure 4.5: Comparison between Original Image and its Version with 2D Projection of corresponding LiDAR points

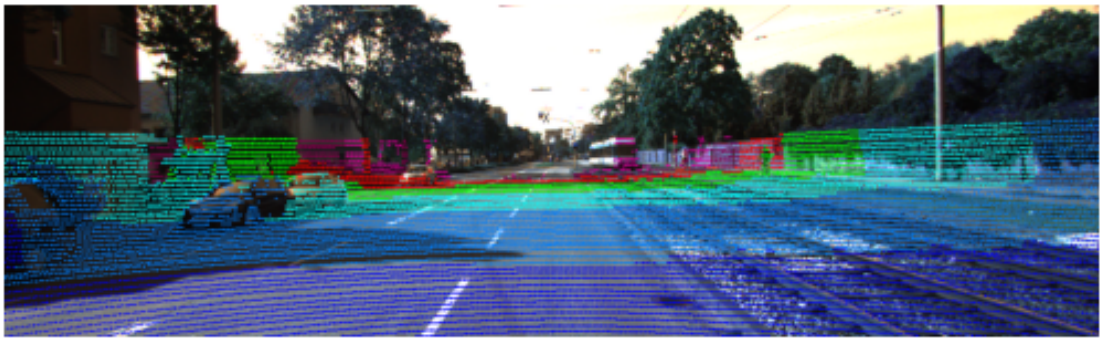

Figure 4.6: Projection of 3D Points into Image Plane

Chapter 5

Concept, Materials and Methods

5.1 Materials

This section provides an overview of the key components and tools used in the research and discuss the materials used, including the advanced sensors and vehicle configuration employed for data collection, as well as the various software tools and libraries utilized for data processing and analysis.

5.1.1 Sensors and Vehicle Configuration for Data Collection

The FZI Research Center for Information Technology's CoCar NextGen, a modified plug-in hybrid Audi A6 Avant aims to improve efficiency, safety, and vehicle communication. With its advanced sensors, powerful hardware, and cutting-edge networking features, this vehicle is ideal for testing connected and AD technologies. This car was built by the FZI with a modular setup, allowing it to adapt to changing research requirements and emerging mobility concepts.

The vehicle is equipped with an array of sensors that are essential for detecting its environment and collecting information for artificial intelligence uses. It's got six 4D LiDAR scanners for detailed 3D mapping, four mid-range 360° LiDAR sensors, and two long-range 360° LiDAR sensors for wide coverage. Plus, there are nine full HD cameras for clear visual info, and three 4D radar sensors to measure speed and distance, even in rough weather. Furthermore, there's a high-precision GNSS system for spot-on positioning, and a Car2X onboard device for real-time communication with other vehicles, infrastructure, pedestrians, and networks.

With this combination of sensors and V2X communication tech, the car's AD and ADAS get a real boost. The car's ADAS system can manage complex traffic conditions, safely change lanes,

maintain the ideal speed and distance from other vehicles, and more by analyzing all the data from various sensors. This doesn't just improve the safety and efficiency of the CoCar NextGen vehicle, it also betters overall traffic flow and safety.

Its modular architecture also allows it to easily incorporate new sensors as they become available, keeping the vehicle at the forefront of AI and AD research. This flexibility is essential for continuously enhancing perception and data production.

Figure 5.1: CoCar NextGen

5.1.2 Software and Tools

A range of software and tools were utilized to facilitate the development and analysis of the research. Python 3.8.10 served as the foundational programming language, providing extensive libraries and support. PyTorch 2.2.2 was employed for DL tasks, leveraging its robust capabilities for building and training NNs. Visualization and plotting of data were achieved using Matplotlib 3.7.5, while CV tasks were handled with cv2 4.9.0. ML models were implemented and evaluated using sklearn 1.3.2. Tensorboard 2.14.0 was instrumented for monitoring and visualizing the training process of NNs. ONNX 1.16.1 facilitated model interoperability by allowing the conversion of models between different frameworks. ClearML was used for visualizing and storing experimental data, ensuring organized and accessible experiment tracking. Additionally, ROS (Robot Operating System) was utilized to access and process data from the CoCar NextGen, enabling integration of hardware and software components in the research workflow.

5.2 Camera data pre-processing and Fault Injection Methods

For our purpose the dataset used is a result of the combination of three separate image datasets such as Canadian Adverse condition Dataset (CACD), KITTI Vision and images taken from CoCar NextGen in the area surrounding the FZI department. The following are the key pre-processing steps applied in the dataset before the injection of faults in the images of an AD car, such that quality and consistency of the data would be maintained. The raw images are collected and reorganized into a structured format, generally in sets of directories that are labeled with categories. Resizing of all the images is usually done into a set of uniform dimensions so a common standard of input is given for fault injection.

The images from different datasets have been adjusted to the same dimensions despite the fact that they were in different dimensions and aspect ratio initially, for achieve same size some parts of the images that were less useful have been cut taking the images to a standard resolution of $512x256$ px. By blending these diverse datasets, it was created a unified collection that reflects various aspects of our surroundings. This process highlights the practical and creative challenges of merging data sources while emphasizing the importance of cohesion in presenting a comprehensive visual narrative. In the process of preparing image datasets for analysis or ML, various pre-processing methods such as color adjustments may be applied. However, it's crucial to consider how these adjustments can impact the consistency and integrity of the dataset. In some cases, applying color adjustments or other pre-processing techniques might introduce inconsistencies or distortions, potentially affecting the reliability of the data, especially in our case.

To mitigate this risk and maintain dataset consistency, a decision was made to opt for a simpler approach: resizing the images while avoiding additional pre-processing steps. Resizing allows for standardization of image dimensions without altering their visual content or introducing potential inconsistencies. By sticking to resizing as the sole pre-processing method, the dataset remains uniform and reliable, facilitating more accurate analysis and modeling.

The dataset construction involved a meticulous process of injecting various types of faults into the initial image dataset, resulting in a comprehensive collection that captures a spectrum of potential imperfections. Each image underwent augmentation by incorporating up to 11 distinct types of faults, strategically introduced in a manner ranging from one to four faults per image. This deliberate approach ensured that the dataset encompassed a diverse array of anomalies, ranging from minor aberrations to more pronounced distortions, effectively simulating real-world scenarios where images may exhibit multiple forms of degradation simultaneously.

By systematically injecting faults across the dataset, we aimed to create a robust training and

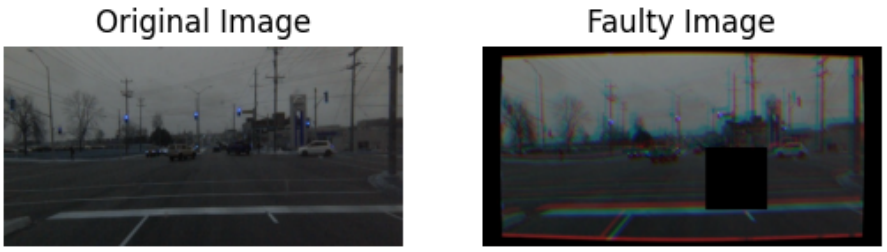

Figure 5.2: Image in different versions taken from the dataset

evaluation resource that not only reflects the inherent complexity of real-world imagery but also facilitates the development and validation of fault detection and correction algorithms under varied and challenging conditions.

Gaussian Noise Injection Injecting Gaussian noise into images involves adding random variations to the pixel values to simulate noise. Gaussian noise, characterized by a mean (in our dataset in a range that goes from -10 to 10) and a standard deviation σ(in a range from 5 to 50), is then generated. This noise follows a Gaussian distribution and is created as a matrix with the same dimensions as the image. The generated noise is added to the original image, and the resulting pixel values are clipped to ensure they remain within the valid range (0 to 255 for 8-bit images). For color images, noise can be added to each channel separately or across the entire 3D array. Finally, the noisy image is either displayed or saved using appropriate functions. This process allows testing the robustness of image processing algorithms and can be used as a data augmentation technique.

Figure 5.3: Example Images from the Dataset with Gaussian Noise Injection

Hot Pixels Injection Hot pixel injection into an image is the process by which bright (often white) pixels are artificially generated to simulate digital camera sensor anomalies. These pixels are often a pre-calibrated number in the image, and the selection is random. The selection is conducted in a probabilistic manner and can be informed on a pixel percentage of the total count or a fixed number. The selected pixels have their values changed to hot pixel values by setting their

values to the maximum intensity, usually 255. This change is done across each channel of color images, such that bright spots are seen. In our implementation the number of hot pixels is in a range that goes from the 1% of the total image area to 3%. The resultant hot-pixel injected image is displayed or saved. This is a useful technique in testing the robustness of image processing or simulating real-world sensor defects.

Figure 5.4: Example Images from the Dataset with Hot Pixels Injection

Gaussian Blur Injection Gaussian Blur can be implemented by convolution on the image by a Gaussian function. A Gaussian kernel is created with a size, normally an odd number such as 3, 5, or 7 (in our case it has been used a range from 3 to 5), and a standard deviation, σ. Using the kernel created provides the amount of blurring; a larger kernel size or higher σ provides more pronounced blurring. The process then applies Gaussian blur on an image by convolution of image pixels with the Gaussian kernel, which averages the pixels with its neighbors, multiplying the averages by Gaussian function. In this way, a smooth image with the image details and edges being blurred is the result.

Figure 5.5: Example Images from the Dataset with Gaussian Blur Injection

Random Rotation Injection A random angle of a certain range can be added to an image to simulate augmentation in a data set. This can be done in a number of ways; for instance, the first step is choosing a random angle within a prespecified range can be generated: in our implementation we decide a range from -20 to 20 degrees, the image can be rotated with this

random angle. A rotation matrix is generated using the center of the image as the pivot. Then, it does apply the matrix in transforming the image. This could involve scaling and translation to ensure the whole rotated image fits into the frame. Thus, the resulting image will be in the form of an image rotated with a randomly chosen angle.

Figure 5.6: Example Images from the Dataset with Random Rotation Injection

Color Shifting Injection In our process, first, the original image is splitted into its color channels: blue, green, and red. Then, each channel gets shifted by some preset amount, moving pixel values along two axes: horizontal and vertical. After that, the pixel values in each channel get clipped to a valid range, so not to overflow or underflow. Then the shifts of the color channels are added back and the color channels combined into a single image. The coloring looks shifted because the alignment of the color channels is deformed in the process, exactly as one would obtain an effect in optical imaging and photographic processes.

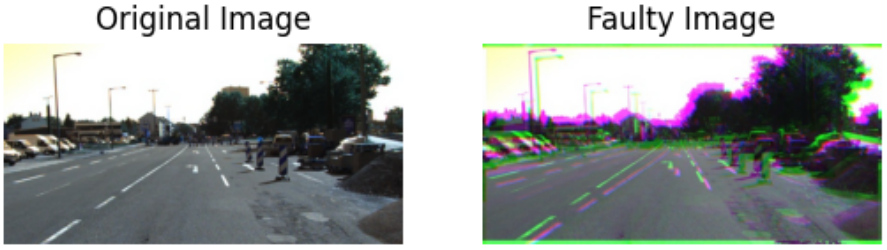

Figure 5.7: Example Images from the Dataset with Color Shifting Injection

Occlusion Injection Occlusions in images hide parts of the image to simulate occlusion of object parts in real-world situations. The process begins with loading an image preprocessed, then a rectangular or irregular region within the image is defined, by selecting a random location and a size of the occlusion in a range that goes from the 1% of the total image area to 5%. This region can be filled with solid colours, noise, or with a patch taken from another area in the image to simulate an occlusion. It effectively occludes a part of the image by manipulating pixel values in

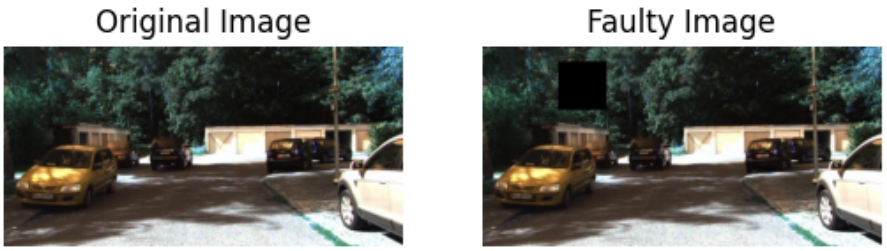

Figure 5.8: Example Images from the Dataset with Occlusion Injection

that region. This is where occlusion techniques get very useful for data augmentation to make ML models more robust by training the model to recognize the object even if a part of the image is lost or hidden.

Lens Flares Injection Artificial light artifacts, emulating the effect of bright light sources interacting with a camera lens, are added to an image to create lens flares. Random lens flares of specific colors are added to arbitrary image positions by simulating the effect of bright light sources interacting with a camera lens. Then, the chosen position for the flare in the image is randomly selected. Often, some shapes such as circles or starbursts are generated with a flare pattern, or a pre-made lens flare pattern is loaded, with its color then adjusted to the desired hue. This pattern is overlayed on the image at the chosen position. The options of intensity and size of the lens flare can be adjusted to the desired visual appearance (in our application the intensity and the number of flares are randomly selected). The technique serves to apply real-looking light effects, and is widely used in photography and graphic design, for giving a more dynamic and engaging composition.

Figure 5.9: Example Images from the Dataset with Lens Flares Injection

Overexposure/Underexposure Injection Injecting overexposure or underexposure into images involves artificially altering the brightness levels to simulate the effects of excessive or insufficient light exposure. To simulate overexposure, the brightness of the image is increased by adding a certain value to the pixel intensities, making the image appear much brighter and potentially

washing out details. Conversely, to simulate underexposure, the brightness is decreased by subtracting a value from the pixel intensities, making the image darker and potentially losing details in the shadows.

Figure 5.10: Example Images from the Dataset with Overexposure Injection

Barrel Distortion Injection In the process that will be described below, barrel distortion is applied to an image, where there is an effect of bending to be outwardly bowed the straight lines in the image when radiating from the center of the image. The process starts off by obtaining the dimensions of an image and proceeds to create a radial distortion map at the center of an image. The radial distance from the center to the pixel is obtained and matched against a distortion formula to attain the curvature in question. And then it takes the new distorted positions of each pixel in polar coordinates, while taking into consideration the strength of distortion previously specified. Then it remaps the pixels with respect to the new coordinates in such a way that the image will be warped so as to exhibit barrel distortion. Such a pixel rearrangement is a simulation of the same effect normally observed with wide-angle lenses. The result is an image with outward-bowed lines, giving a very realistic simulation useful for use in a myriad of applications, from image processing to visual effects.

Figure 5.11: Example Images from the Dataset with Barrel Distortion Injection

Pincushion Distortion Injection The scope of this process is to distort the image so it has a pincushion effect much the same as can be seen with some lens setups. In the first phase, the

distance of each pixel in an image to the relative center is calculated. Then the degree to which each pixel is at a distance from the center is increased by some user-defined amount. What is done after is virtually push the image data outward evenly, just as would the inward curve of a pincushion. The changed distances are used to remap the complete image, that remapping actually brings in any straight lines, thus forming the typical pincushion distortion. The degree of the distortion effect can be controlled by using the before mentioned intensification value.

Figure 5.12: Example Images from the Dataset with Pincushion Distortion Injection

Jitter Injection Injecting jitter into images involves introducing random displacements to the pixel positions to simulate the effect of camera shake or slight movements. For each pixel or a block of pixels in the image, a small random shift is applied both horizontally and vertically. These shifts are determined by generating random values within a specified range, ensuring that the displacements are subtle and maintain the overall structure of the image. The pixels are then moved to their new positions based on these random shifts. After applying the jitter, the resulting image appears slightly blurred or jittered, mimicking the effect of minor, rapid movements during the capture. This technique is useful for data augmentation, helping to make ML models more robust by training them to recognize objects even when there is slight motion blur or displacement.

Figure 5.13: Example Images from the Dataset with Jitter Injection

5.3 Fault Detection Using Traditional techniques

Traditional fault detection in images is an old concept that involves many state-of-the-art methods applied both industrially and in research settings. The basic methods for identifying faults in images filtering, edge detection, thresholding, and morphological transformations are based on the most fundamental operations of image processing. The filters, such as the Gaussian and median filters, cut the noise in order to make the faults clearer. Edge detection methods, such as Sobel, Canny, and Prewitt, are necessary for marking discontinuity and image anomalies. These are needed for faults expected in the image. Grayscale to binary conversion is implemented both by global and adaptive thresholding, contributing greatly to the isolation of regions of interest and making it easier to recognize defects in such images. Morphological transformations, such as dilation and erosion, enhance the structural features of the determined faults, therefore making them more perceivable. Additionally, statistical analysis methods like histogram-based approaches or image differencing are employed to detect anomalies, changes, or degradation in image quality. Meanwhile, in LiDAR systems, fault detection traditionally revolves around signal processing methodologies, focusing on analyzing the return signals and evaluating deviations from expected patterns. Time-of-flight measurements, signal amplitude analysis, and data coherence checks are some techniques used to identify faults like sensor misalignments, corrupted data points, or erroneous measurements.

Identifying and diagnosing faults, such as dead pixels, lens flares, and color inconsistencies, is mainly the domain of traditional image processing techniques. They involve a series of methodical steps, comprising image preprocessing, segmentation, and feature extraction, to detect anomalies that might point to sensor malfunction. With the transformation of images into different color spaces, thresholding, and morphological operations, these techniques could effectively highlight defects that are difficult to spot by other means. In the next few pages will be examineed the application of traditional image processing techniques for the detection and analysis of faults in camera sensors.

5.3.1 Gaussian Blur detection

Gaussian blur detection in images is a critical aspect of image processing, particularly in applications requiring clarity and precision. Gaussian blur, a type of image smoothing, reduces image noise and detail by averaging the pixel values with their neighbors weighted according to a Gaussian distribution. Detecting this blur is essential in fields such as medical imaging, surveillance, and digital photography to ensure image quality and integrity. Techniques for Gaussian blur detection, such as [69], often involve analyzing the frequency domain of images or using spatial domain methods

to measure the smoothness or gradient of pixel intensities. For instance, the Fourier Transform is commonly employed to identify the frequency components altered by blurring. Additionally, edge detection algorithms [44], such as the Canny or Sobel operators, can be adapted to detect the loss of sharpness indicative of Gaussian blur. Recent advancements incorporate ML models trained to recognize blurring patterns, providing higher accuracy and robustness.

The implemented method for blur detection begins with reading an image and converting it to grayscale, which simplifies the subsequent processing steps. Following this, the variance of the Laplacian of the image is calculated, as it serves as a robust measure for detecting blur. The Laplacian, defined as

$$\mathcal{L}(\mathcal{I}) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2},$$

accentuates areas of rapid intensity change, highlighting edges and details. The variance of these Laplacian values,

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(L_i - \mu)^2,$$

where L_i represents the Laplacian value at pixel i, μ is the mean of the Laplacian values, and N is the total number of pixels, provides a quantitative measure of the image's sharpness. If this variance is below a predetermined threshold, the image is classified as blurred. This method is computationally efficient and effective across various types of blur, demonstrating high accuracy in empirical tests [7, 30, 43].

5.3.2 Hot pixels detection

Hot pixel detection in images is one of the important activities in digital imaging and has a general application for the improvement of photographic images with low illumination or long exposure. A hot pixel is a single pixel of the image sensor that is much more illuminated than the surrounding pixels due to manufacturing imperfections, high temperatures, or overexposure. Classic methods, [26], for detecting hot pixels are based on the analysis of pixel intensity values in several frames. Some methods are based on statistical ones that find outliers peculiarly extremely bright. Advanced methods, such as [63], exploit image processing using techniques and algorithms like median filtering and thresholding to accurately isolate hot pixels.

The process implemented starts filtering the grayscale image by a threshold to pick out those hot pixels, creating a little mask for them. Next up, the outlines are searched, and the system starts to count how many of these outlines fit certain conditions. For each outline, it is calculated a region of interest (ROI) around the hot pixel and figure out the average intensity of the pixels

within that area. On top of that, it is considered the intensity of the pixel in the middle of the ROI, and the difference between the average intensity and the center pixel's intensity. If the area of the outline is less than a set threshold and the intensity difference is more than a certain limit the pixel is classified as hot pixel.

5.3.3 Gaussian Noise detection

Gaussian noise occurs due to the inherent shortcomings of sensors or electronic disturbances at the instant of imaging, and it is expressed as randomly varying pixels in intensity, distributed according to a normal distribution. This detection involves statistical analysis of the pixel intensity properties and a deviation from the patterns. Filtering is applied to distinguish Gaussian noise from the real image content.

The algorithm first converts the image into grayscale, then, it blurs the image using a Gaussian blur filter to reduce noise. At this point the mean squared error of the difference between the original image and the denoised image is computed according to the formula:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2$$

If the MSE is not zero the PSNR is computed from the given equation:

$$PSNR = 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right)$$

Now, MAX_I represents the maximum possible pixel value of the image. Then, the algorithm evaluates the quality of the image by returning True when the PSNR is less than a threshold, which implies that the image is too noisy; otherwise, it returns False. This procedure will help in the proper evaluation of the noise in the image, after which further image processing can be carried out.

5.3.4 Rotated images detection

Rotated images detection is a task in CV, particularly interesting for applications involving image recognition, object detection, and automated image processing. This process involves identifying images that have been rotated from their original orientation and determining the angle of rotation. Advanced algorithms and techniques, such as Hough transforms [18], and feature matching, are commonly employed to detect and correct rotated images. These methods analyze the visual patterns, edges, and keypoints within the image to infer its correct orientation.

Proposed algorithm starts applying a threshold to the grayscale image to get a binary image for easy contour detection. After having the binary image ready, the algorithm identifies the contours available in it. Subsequently, it checks if these contours are near the edges of the image. If a contour is observed close to the border and its area is above a certain minimum value by default, this flags the image as rotated. This ensures that features near image borders, significant in size, would be detected and used as markers of the possible rotation of the image to continue in further correction and processing.

5.3.5 Color shifting detection

Shifts can be caused by a great many number of things: lighting, different camera settings, errors of digital processing, and so on. Detection of color shifts can be done by comparing the color values in the image with a known standard or a reference image. Common approaches for this are histogram analysis, color space transformation and Fourier Transform pattern detection. Technically, exact detection of shifts in color is crucial for most applications which are related to preservation of visual information where the requirements for visual fidelity and consistency of images are established. The algorithm described below is based on the recognition of a certain pattern in the Fourier Transform of the input image given by the artifacts produced injecting color shift in images. The proposed algorithm begins by executing the Fourier Transform on the grayscale image to calculate its magnitude spectrum. This spectrum is then normalized and converted into a grayscale image for further processing. The subsequent step involves applying an adaptive thresholding method to the normalized magnitude spectrum image. Following thresholding, two morphological operations are performed: an opening operation to remove noise and a closing operation to fill gaps. The morphological operations can be defined as follows, according to 3.3.3:

$$\text{Opening: } (A \circ B) = (A \ominus B) \oplus B$$

$$\text{Closing: } (A \bullet B) = (A \oplus B) \ominus B$$

where $\ominus$ and $\oplus$ denote the erosion and dilation operations, respectively.

Next, the Canny edge detection algorithm is applied to the morphologically processed image to detect edges. The Canny edge detection process involves the following steps: applying a Gaussian filter to smooth the image, finding the intensity gradients, applying non-maximum suppression, and using double thresholding to detect edges. Finally, the Hough Transform is utilized to detect lines in the edge-detected image. The Hough Transform identifies lines by transforming points in

the Cartesian coordinate system to the parameter space. The algorithm counts the number of lines that have angles between 10 and 70 degrees. This count determines the classification of the image based on a predefined threshold of 10 lines. The angle θ is measured from the horizontal axis, and the number of lines within this range is used as a metric for classification purposes. In fig 5.14 there is an example of classification.

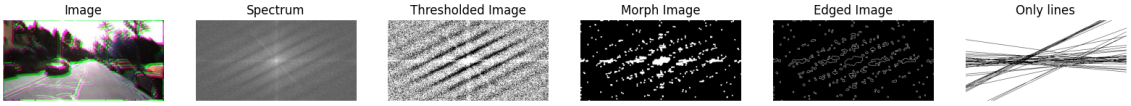

Figure 5.14: Pipeline of the process used to detect color shift, the original image is firstly inspected in Fourier space, then a threshold process and a morph filter is applied, the result edged image can be therefore processed by a Hough transform to get the lines.

5.3.6 Occlusions detection

Occlusion detection in images is an important task in many CV applications. The methodology for occlusion detection consists of a series of key steps: starting with the grayscale transformation of the image, thresholding is then used to further reduce the grayscale image to a binary image, which gives a way of distinguishing the background from the foreground or potential occlusions based on pixel intensity values. The binary image is inverted to flip the foreground and background to simplify the detection of potential occlusions. From the inverted binary image, contours are identified with contour detection algorithms. Each contour detected is then analyzed to see if it constitutes an occlusion. More specifically, it requires checking whether the area of the contour is greater than some minimum threshold and it should not be too close to the border of the image. Thus,

$$\text{Area}(C) > A_{min} \quad \text{and} \quad \text{distance}(C, \text{border}) > d_{min}$$

where $\text{Area}(C)$ is the area of contour C, A_{min} is the minimum area threshold, and d_{min} is the minimum distance from the image border. If a contour passes these criteria, then an occlusion is detected.

5.3.7 Lens flares detection

It is considered very important to detect lens flare images to understand how reliable and precise CV systems are. Lens flares from bright light sources, like the sun, in camera images can seriously affect the way the surrounding world is perceived by autonomous vehicles. These artifacts may result in potential misidentifications of an object or a road situation.

In the proposed technique, firstly the image is converted to the HSV color space. This is done because working with colors in the HSV space is often more intuitive and effective for color-based segmentation compared to the RGB color space. Two bounds that represents the lower ([0, 0, 200] in RGB space) and upper ([0, 0, 200] in RGB space) bounds for the yellow color in the HSV space are then defined, these bounds are used to generate a binary mask of the image where pixels inside the yellow color range are set to white and all other pixels are set to black. The yellow mask is then subjected to two morphological operations: opening and closing, using an elliptical structuring element. These operations refine the mask for better detection of possible lens flares. After these morphological operations, contours in the mask are found. Then, for every contour found, the code calculates an area and perimeter; after that, it calculates the circularity, which is a measure of how close the contour is to a perfect circle, if the circularity of any contour is higher than a threshold (0.8), it means that the image there is supposed to have a lens flare.

5.3.8 Barrel distortion detection

Detecting and correcting image distortions such as pincushion and barrel distortions is vital for accurate object detection and navigation. Barrel distortion, common in wide-angle lenses, makes images appear to bulge outwards at the center, while pincushion distortion, more typical in zoom lenses, causes images to pinch inwards towards the center. These distortions can severely affect the accuracy of visual perception systems in autonomous vehicles. The algorithm described for image analysis attempts to detect specific features in an image through a sequence of well-defined image processing steps. The image is firstly converted to grayscale This conversion is a key step for the later binary thresholding operations: The grayscale image is so converted into a binary image. Then, the colors of the binary image are inverted to ensure the regions of interest are white and correctly identified for the morphological processing. The closing morphological operation is then applied to the inverted binary image to remove non-circular parts and fill small holes. The closing operation is done by creating a kernel and applying it the closing operation. The algorithm then counts the number of white pixels in the top, bottom, and side regions of the image to measure the extent of the white regions, which may be indicative of occlusions or artifacts. Finally, the distortion is detected if the sum of the white pixels exceeds a percentage threshold (3% of the image area) of the total number of pixels in the image, which is indicative of the feature being present.

5.3.9 Other faults detection

While this techniques form a real tool in fault detection for camera sensors, several limitations must be drawn from traditional image processing techniques. To that end, some kinds of faults like pincushion distortions, overexposure, underexposure, and jitter can be problematic to several image-processing techniques on account of their complexity and subtlety. Fault variability and unpredictability can yield low accuracy of several algorithms for fault detection. Future improvements in image processing methodologies help in remedying the present limitations and improving fault detection systems for camera sensors. On this note, current research in emerging technologies such as DL and CV aims to help this research overcome the pitfalls and ultimately come up with a more resilient solution for making imaging systems more reliable and performing well in the broadest range of applications.

5.4 Deep Learning Techniques for Fault Detection

For this part of the thesis a great effort is made for training up such a supervised deep-learning model to classify the input image into one of the following 12 fault classes: One class for images without faults, and other 11 classes for each specific fault, requires a very comprehensive and careful evaluation of quite a few advanced architectures including some of the most promising models, for this scope, in our research we took in consideration architectures such as ResNet50, INceptionV3, and EfficientNet.

We will also introduce a two-tiered strategy that combines these models to maximize performance and computational efficiency. The first tier employs a lightweight model like EfficientNet for initial classification, while the second tier uses deeper architectures like ResNet50 for more complex fault identification. An overview of techniques used in this chapter is offered in fig. 5.15. While this section will be based on the concepts and implementation, the performance of each model will be summarized in terms of accuracy, precision, recall, F1-score, computational time and inference time in the evaluation section.

5.4.1 Comparative Analysis of State-of-the-Art Models for Fault Detection and Classification

Most of the models have been evaluated in the development of a technique for the detection of errors in images to understand their training times, inference times, and performance in general. The synergy of such an evaluation is aimed at understanding what models are most efficient and

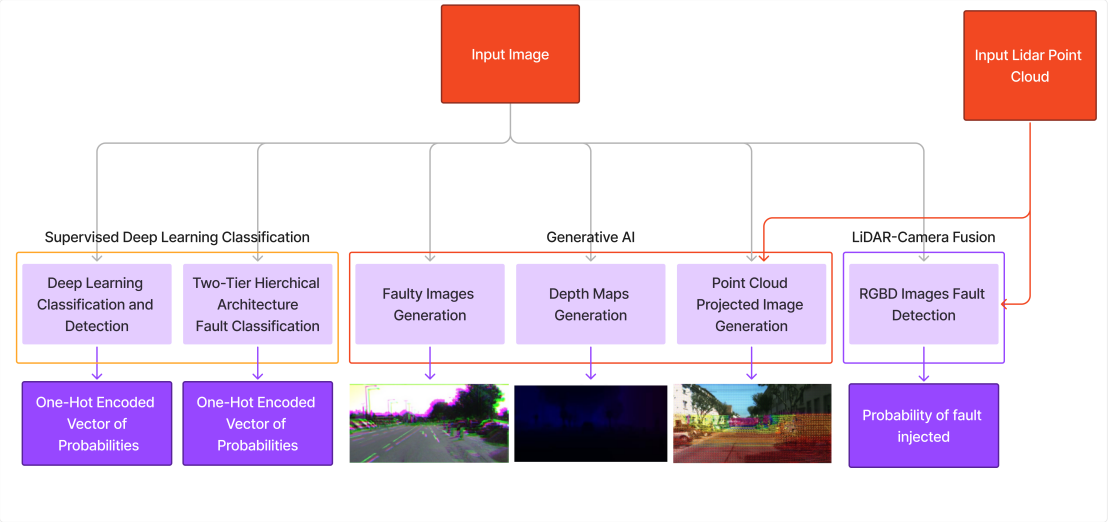

Figure 5.15: Flowchart that outlines a workflow that combines supervised deep learning, generative AI, and LiDAR-camera fusion for fault detection and image generation. The process starts with an input image, which is processed through supervised deep learning classification using deep learning models for fault detection and a two-tier hierarchical classification, both providing a one-hot encoded vector of probabilities. Generative AI is then used to create faulty images, depth maps, and point cloud projections, enhancing data diversity and training. Finally, LiDAR-camera fusion integrates LiDAR and camera data for improved fault detection in RGBD images, resulting in the probability of fault presence.

accurate in their performance for our purposes. First, consideration and testing were estimated over a robust dataset for training with many recently proposed parameters and techniques that are applied before measuring with some state-of-the-art models. Many of them are in the list of the most current advanced architectures. Then, after this preliminary comparison, each model underwent over a training process augmented with a series of refinements designed to optimize performance. These were critical in taking the models from their baseline capabilities to models that can provide the needed high accuracy and efficiency demanded by our task. An implementation of a learning rate schedule was done so that the learning rate could be adjusted dynamically during training. In addition, a weighted loss function was used, based on the frequency of classes in the dataset, so that less frequent classes were given appropriate emphasis during training. Class weights were used to further balance the effect of each class on the overall training process. In the next pages each of these techniques will be described in order to understand the benefits brought to the models performance. In order to be able to have a uniform output with respect to the number of classes, the changes made mainly concern the last layer, which has been modified so that the output of the models is a vector of 12 elements that for each value represents the probability that the input image belongs to a class. In other words, the $i-th$ value of the vector will represent

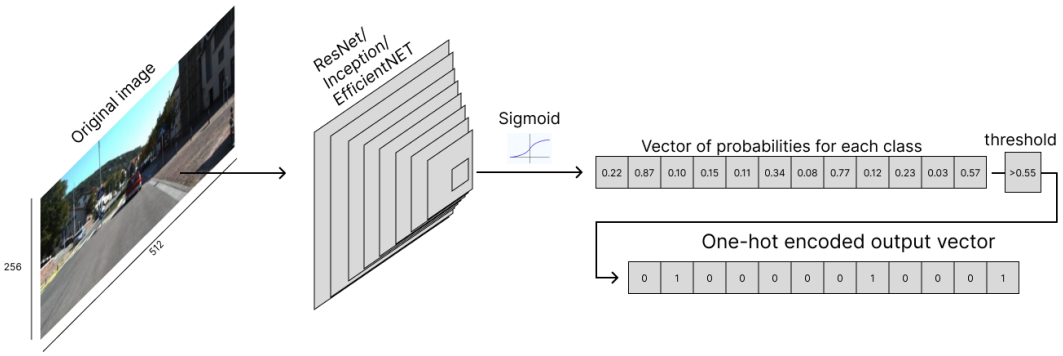

Figure 5.16: This diagram depicts the fault classification process using various deep learning models such as ResNet, Inception, and EfficientNet. An original input image is processed through one of these models, which outputs a vector of probabilities for each fault class. A sigmoid function is applied to convert the raw outputs into probabilities. A threshold is then applied to these probabilities to generate a one-hot encoded output vector, indicating the detected faults in the image based on the predefined classes.

the probability that the input belongs to the $i-th$ class, of course an image could belong to more than one class, so the final output should be evaluated using a threshold after applying a Sigmoid function. A visualization of the architecture is offered in fig. 5.16. For the training the loss function used is the Binary Cross Entropy Loss, in particular the implementation provided by PyTorch, [16], called BCEWithLogitsLoss. This loss combines a Sigmoid layer and the BCELoss in one single class. This version is more numerically stable than plain Sigmoid followed by BCELoss as by combining the operations into one layer, we can take advantage of the log-sum-exp trick for numerical stability.

Binary Cross entropy loss defines the difference between the probable predictions and the actual binary-type labels. The output is in the form of a probability value representing class membership in modeling in NNs. The unreduced (i.e., with reduction set to 'none') loss can be described as:

$$\ell(x, y) = L = \{l_1, \dots, l_N\}^\top,$$

$$l_n = -w_n \left[y_n \cdot \log \sigma(x_n) + (1 - y_n) \cdot \log(1 - \sigma(x_n)) \right],$$

where N is the batch size. If reduction is not 'none' (default 'mean'), then

$$\ell(x, y) = \begin{cases} \text{mean}(L), & \text{if reduction} = \text{'mean'}; \\ \text{sum}(L), & \text{if reduction} = \text{'sum'}. \end{cases}$$

Using this weighted loss we can inject class-weighting, which is one of the ways to cope with

imbalanced datasets by providing more emphasis on the minority class.

This type of weighting is typically applied according to a class distribution in the dataset, ensuring that contributions by the misclassification of the minority class are reasonably high and guide the model with more attention towards learning representations from minority data. Therefore, the weighted loss function usually becomes a powerful tool to improve the performance of models on imbalanced datasets by addressing bias towards the majority class. The weights point in the direction of a class imbalance that is pretty significant, with specific class like "No fault" having moderately higher weights than the others. This, by all means, sends a strong message that some faults are either more predominant or are maybe more of a critical issue within the dataset than others. To do so, the classes were assigned weights so that the model prediction isn't biased firmly towards one of the faulty types, which are more frequent. Such a weighting method helps to approach a balanced learning process, giving roughly equal attention to all classes during training. By properly weighting classes, the model becomes more robust and accurate in identifying less frequent but crucial faults hence, better overall performance and reliability in practice. The visualization 5.17 is helpful in demonstrating how class weights are spread across the different types of faults in a dataset.

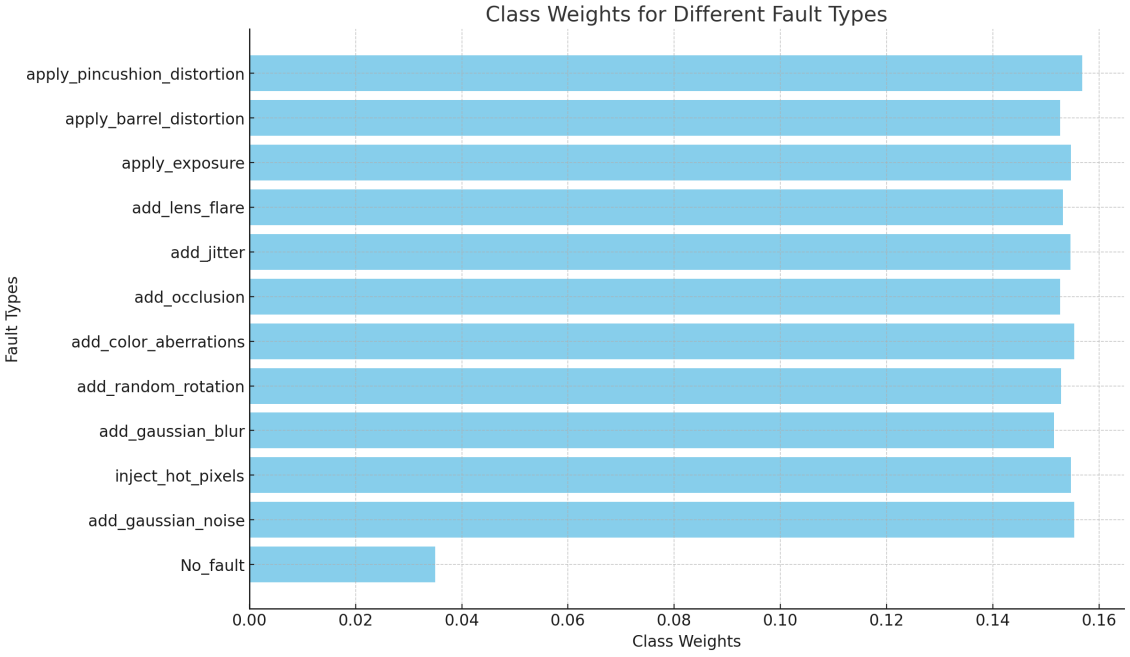

Figure 5.17: Bar Plot that describes weights for each class used for calculation of the loss

It is also possible to trade off recall and precision by adding weights to positive examples. In the case of multi-label classification, the loss can be described as:

$$\ell_c(x, y) = L_c = \{l_{1,c}, \dots, l_{N,c}\}^\top,$$

$$l_{n,c} = -w_{n,c} \left[p_c y_{n,c} \cdot \log \sigma(x_{n,c}) + (1 - y_{n,c}) \cdot \log(1 - \sigma(x_{n,c})) \right],$$

where c is the class number ($c > 1$ for multi-label binary classification, $c = 1$ for single-label binary classification), n is the number of the sample in the batch, and p_c is the weight of the positive answer for the class c.

$$p_c > 1 \quad \text{increases the recall,} \quad p_c < 1 \quad \text{increases the precision.}$$

For example, in our implementation the positive weights have been calculated trough a refinement process based on successive iterations. The first iteration was used to understand in which classes there was a lack of performance, the main problem was with the recall due to an high rate of images classified as false negatives, so after the first results weights were calculated based on the first recall values.

$$W_i = \frac{1}{Recall_i}$$

Where i indicates the index of the class. In this visualization 5.18, we present a horizontal bar plot that depicts the positive weights assigned to various faults used in the training of a ML model.

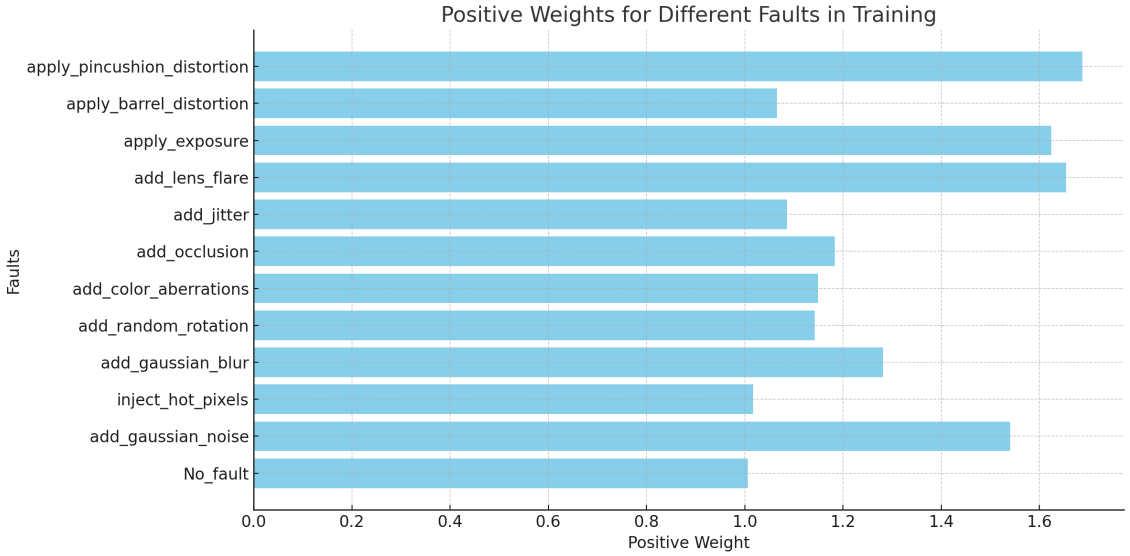

Figure 5.18: Bar Plot that describes positive weights for each class used for calculation of the loss

Starting from the baseline the first upgrade made to improve the learning process is about the

learning rate scheduler. The first scheduler used was based on a constant learning rate of 0.005, in this kind of configuration the learning rate should be chosen wisely and this leads to an high amount of experiments to choose the right value that cannot also be changed dynamically during the training. In our implementation some schedulers have been tried, such as linear, constant and exponential schedulers, but the best performance were given by the cosine annealing scheduler. To briefly introduce the schedulers mentioned before: Linear annealing reduces the learning rate linearly over time, following the formula

$$\eta_t = \eta_0 - \frac{t}{T}(\eta_0 - \eta_{min})$$

where η_t is the learning rate at epoch t, η_0 is the initial learning rate, T is the total number of epochs, and η_{min} is the minimum learning rate.

Constant annealing maintains a fixed learning rate throughout training, expressed as $\eta_t = \eta_0$, ensuring steady updates.

Exponential annealing decreases the learning rate exponentially according to

$$\eta_t = \eta_0 \cdot \gamma^t$$

where γ is a decay factor.

Cosine annealing modulates the learning rate using a cosine function, described by

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_0 - \eta_{min}) \left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$$

allowing for periodic reductions that can help escape local minima. In fig. 5.19 a graph representing the functions describing the learning rate trend with values used in training is presented.

Key components and configurations would be fed into the training process of the machine-learning model so optimal performance and learning can be achieved. The key elements are enlisted below for discussion under the training process.

Training is under the optimizer AdamW [42]. AdamW is an adaptive learning rate optimization algorithm, and the primary objectives include the combination of the advantages of Adam and weight decay. This optimizer may enable high effectiveness in coping with problems having sparse gradients in noisy issues, so it is very suitable for most DL tasks. To further reduce overfitting and promote generalization, weight decay is included in the model.

The batch size used is 32, meaning that 32 samples are being processed in the model before updating the model parameters. The batch size of 32 is used so that the computational efficiency

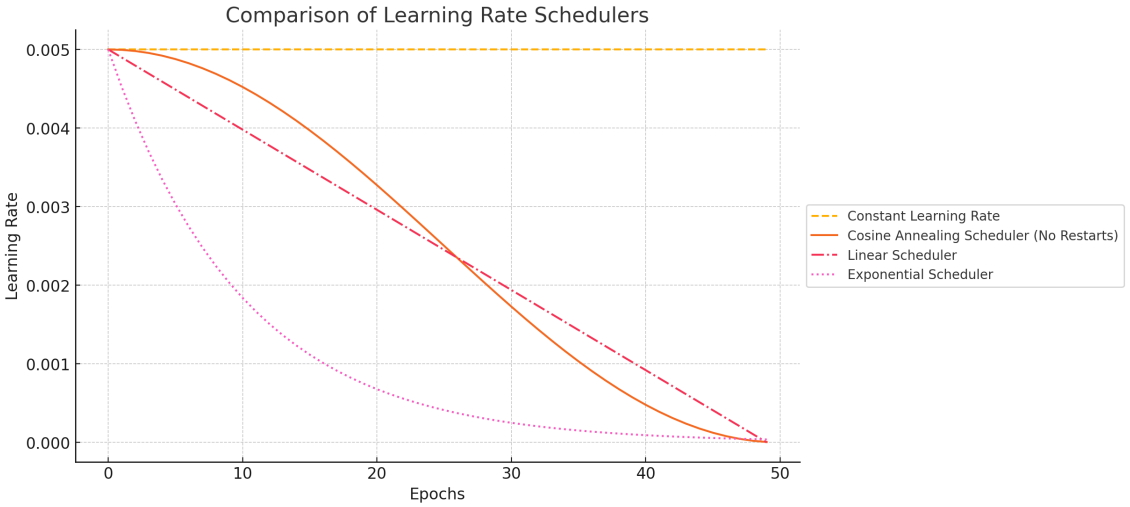

Figure 5.19: This plot illustrates the changes in learning rates over 50 epochs for four different learning rate schedulers: Constant, Cosine Annealing (without restarts), Linear, and Exponential. The Constant scheduler maintains a fixed learning rate, while the other three schedulers decrease the learning rate according to their specific strategies, showing different rates and patterns of decay.

of the model is not too degraded while, at the same time, preserving the stability of the updates of information in the gradients, which is indispensable for the model's convergence.

The training process was carried over epochs ranging from 30 to 50. An epoch is counted as one complete pass through the whole training set. Training during multiple epochs allows the model to learn and tune its weights progressively, thereby improving the task of generalizing from the training data to unseen data.

The evaluation of the architectures mentioned before helps outline their strengths and weaknesses in image classification performance and computational efficiency, comparing them on the same task with almost same training parameters could give a good overview of the performance for each of them.

The first model evaluated is *ResNet50*, this model happens to be a type of Residual Network (ResNet) model architecture that is notable for the innovative use of residual connections. ResNet50, introduced by He et al. [25], is a 50-layer deep network that allows the gradients to flow through the shortcut connections directly, making the vanishing gradient problem a thing of the past. This can train a far deeper network than is possible with a regular CNN and with much better accuracy. ResNet50 is made up of many residual blocks, in which there are convolutional layers followed by batch normalization and then ReLU activations. Subsequently, this structure forms a complementary network that enhances feature learning and makes the network more robust to degradation with increased depth. Then, *InceptionV3*, proposed by Szegedy et al. [57] as well,

is part of the Inception family and combines the idea of factorized convolutions with aggressive dimension reductions to make structures computationally efficient without compromising model performance. In contrast, InceptionV3 builds a modular architecture in which each module, or block, attempts to capture a different spatial correlation and level of abstraction through a group of parallel convolutions of varying sizes. This way, the network creates a multipath way to embed more expressive features in the network. Next, methods are applied in training the network, such as batch normalization, label smoothing, and optimization using RMSprop, to enhance overall efficiency in training and model accuracy.

Finally, *EfficientNet B0* [58] and *EfficientNetV2* [59] roposed by Tan and Le, are both convolutional NN architectures designed for image classification, but they differ significantly in their design and performance. EfficientNet B0 is the baseline model of the original EfficientNet series, known for its balanced compound scaling method that optimizes depth, width, and resolution for efficiency and accuracy. Using the knowledge of the compound scaling procedure that was introduced in the original paper of EfficientNet, EfficientNetV2 ensures that the depth, width, and resolution are uniformly scaled through fixed scaling coefficients. The tiny variant of EfficientNetV2, on the other hand, is optimized for maximal speedup in training, parameter efficiency via progressive learning in which the model progresses both in input image size and in network depth and achieves high accuracy while demanding less computation and memory compared to other architectures with matching performance.

5.4.2 Hierarchical 2-Tier Architecture for Fault Detection

As stated before, this model was used to come up with a two-tier ML model for robust fault detection with efficiency and deep analytical abilities. It uses a very efficient classifier at first, in our case EfficientNet B0, this decision was made due to EfficientNet's computational efficiency and capability in image recognition problems, resulting in a model that can quickly inspect passing images and highlight potential faults. Streamlined architecture of EfficientNet ensures fast processing of data when dealing with detection of anomalies.

When a given image has been flagged by the first-pass classifier as potentially faulty, it is then passed on to the second stage of our model for further inspection. At this stage, a larger, more complex model, eg. ResNet50 is used. We use ResNet50 because it is a more advanced architecture that can work well with fine features and detailed data patterns. This facility helps in correctly recognizing and diagnosing the nature of the faults, thus providing a more detailed view of the identified defects. A visualization of the proposed architecture is offered in fig. 5.20. For this

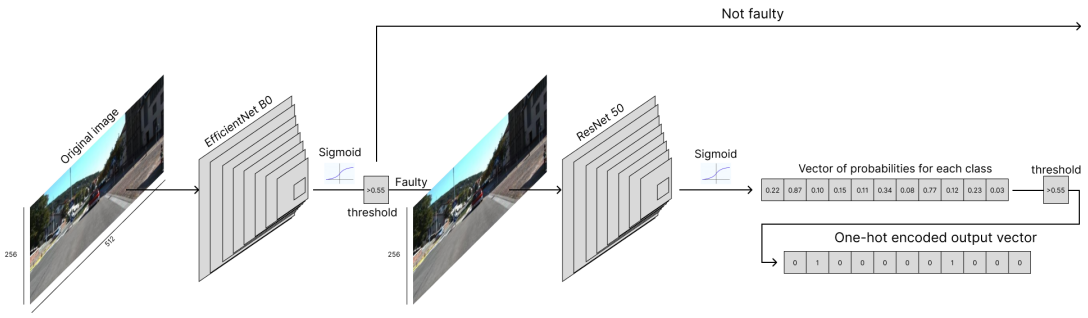

Figure 5.20: This diagram illustrates a two-tier classification framework for detecting and classifying faults in images. The first tier utilizes an EfficientNet B0 model to perform initial classification, determining whether an image is faulty or not based on a threshold applied to the sigmoid output. If a fault is detected, the second tier employs a ResNet50 model to further analyze the image, producing a vector of probabilities for each fault class. A threshold is then applied to convert these probabilities into a one-hot encoded output vector, indicating the specific type of fault detected.

architecture, we decided to be consistent by using virtually the same settings as were used in the application described in the preceding section. It consists of the same starting learning rate, the size of the batch, the loss function, and the optimizer.

5.5 Generative-AI

Then, we will discuss the use of image-to-image generative models, specifically the Pix2Pix architecture, to produce images with injected faults. This approach leverages the model's ability to learn mappings between normal and fault-injected images, allowing for the automatic generation of diverse and realistic faulty images. The Pix2Pix model is trained on a paired dataset consisting of normal and fault-injected versions of the same images, which enables it to translate normal images into their faulty counterparts.

This methodology enhances the dataset by creating a wide variety of images, including both expected and unexpected faults, thereby improving the robustness of downstream models. Additionally, by training on these paired datasets, the Pix2Pix model can learn to generate novel fault scenarios that were not part of the original training data, increasing the diversity and completeness of the datasets.

5.5.1 Generation of Faulty Images Using Generative Models

As stated in the previous section the architecture used to reach the goal of expanding the dataset is the Pix2Pix architecture, introduced by Isola et al. [29]. This architecture represents a decisive innovation in image-to-image translation, utilizing a generative adversarial network (GAN) frame-

work. This architecture excels in transforming input images into corresponding output images, facilitating a range of CV tasks.

Pix2Pix comprises two core components: the generator and the discriminator, working in a conditional GAN setup to produce high-quality images from input images.

The generator employs an *encoder-decoder* structure, following a U-Net architecture in which the encoder consists of a series of convolutional layers that downsample the input image, capturing spatial features at various scales. Each layer integrates convolution, batch normalization, and Leaky ReLU activation functions. The bottleneck layer, located at the lowest resolution, captures the most abstract features of the input image. The decoder mirrors the encoder and uses upsampling to reconstruct the image, with transposed convolutional layers increasing spatial resolution. Each layer includes transposed convolution, batch normalization, ReLU activation, and dropout layers to prevent overfitting. Skip connections between corresponding layers of the encoder and decoder preserve spatial information, enhancing detail and accuracy in generated images. The discriminator functions as a PatchGAN, classifying each 70x70 image patch as real or fake, which sharpens the focus on high-frequency details. It comprises:

- **Convolutional layers** downsample the input image or patch, extracting hierarchical features.
- **Batch normalization** and Leaky ReLU stabilize training and improve convergence.
- The **final classification layer** outputs a probability map indicating the likelihood of each patch being real or fake.

Pix2Pix training utilizes two loss functions: The first one is adversarial Loss that encourages the generator to produce realistic images:

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

where G is the generator, D is the discriminator, x is the input image, y is the ground truth image, and z is the noise vector. The L1 Loss ensures the generated image closely matches the ground truth in pixel space:

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

The generator's total loss is a weighted sum of the adversarial loss and the L1 loss:

$$\mathcal{L}_G = \mathcal{L}_{GAN}(G, D) + \lambda \mathcal{L}_{L1}(G)$$

with λ balancing the two losses.

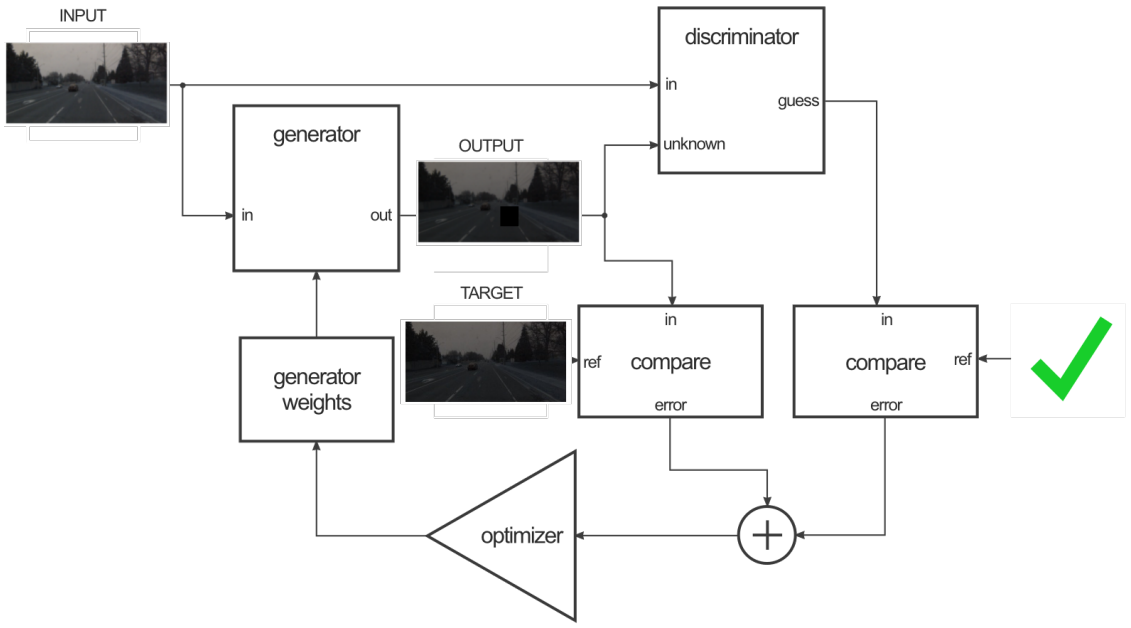

Figure 5.21: Training process of the pix2pix GAN architecture for image-to-image translation. The diagram illustrates the interplay between generator and discriminator networks, showcasing the flow of input, output, and target images through the system during the training phase. Image adapted from [29].

The parameters utilized to train the model model are almost the same as those proposed in the original paper. The initial learning rate is set to $2e^{-4}$ with a batch size of 16, which helps in stabilizing the training process. The model is trained for 300 epochs to ensure thorough learning. The Adam optimizer is used with $\beta_1 = 0.5$ and $\beta_2 = 0.999$ to efficiently handle sparse gradients. The loss function combines the L1 loss, with $\lambda L1$, a weighting factor for the L1 loss that ensures the generated images are close to the ground truth in pixel space, set to 100, and the adversarial loss, promoting pixel closeness and realistic image generation. Notably, λGP, which is often used for gradient penalty in GANs to improve training stability, has not been used in this setup. The training result are showed in fig. 5.22, it is interesting to notice how during the training the model starts to learn new types of faults and how to inject them into the images.

5.5.2 Generation of Depth Maps Using Generative Models

To generate depth maps using our dataset, we leveraged the principles outlined in the *UniDepth* architecture proposed by Piccinelli et al. [48], which focuses on universal monocular metric depth estimation (MMDE). This process involves using a single image to predict dense metric 3D points

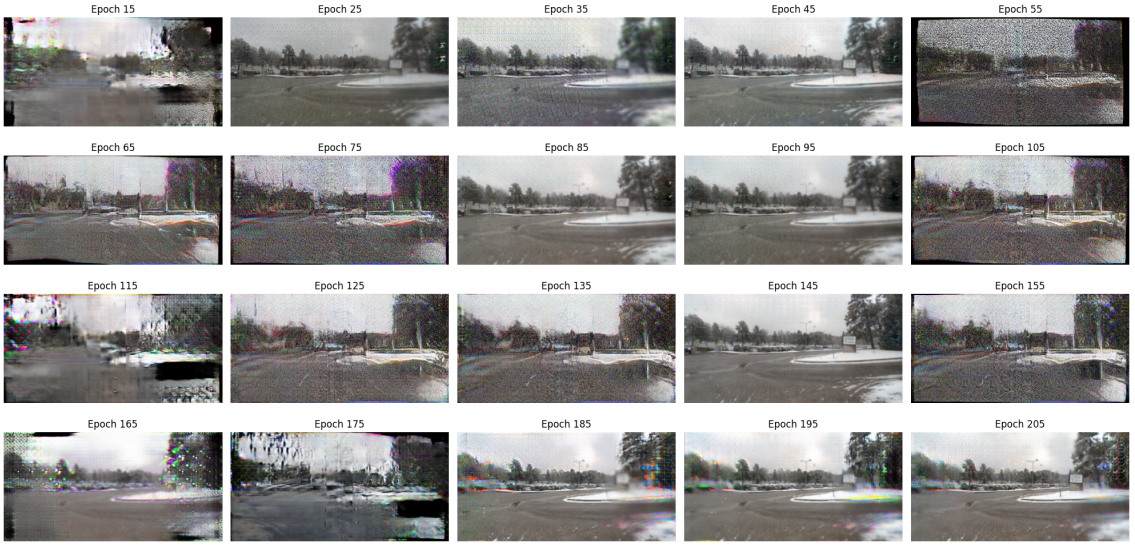

Figure 5.22: Progressive learning stages of the pix2pix framework for fault injection in images. The sequence illustrates the model's output from epoch 15 to 205, demonstrating the gradual refinement in synthesizing realistic fault effects.

without requiring additional information such as camera parameters. UniDepth incorporates a self-prompting camera module that predicts dense camera representations to condition depth features effectively. This method employs a pseudo-spherical output space to disentangle camera and depth representations, enhancing the model's ability to generalize across different domains.

We used pre-trained models for the whole architecture. These were then used directly to test their prediction quality against our dataset of RGB images, with LiDAR-generated depth maps. This pre-training helped importantly in giving a very strong initialization where on their own, powerful predictions on our data alone were possible without any further fine-tuning. A result of the testing can be found in 5.23.

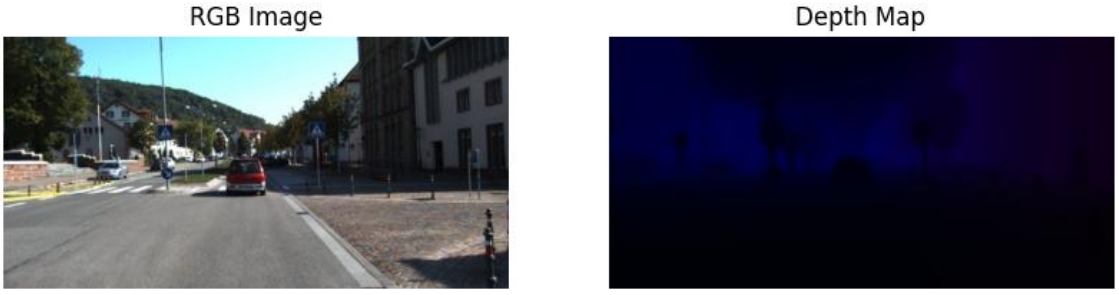

Figure 5.23: Figure displaying RGB image alongside its corresponding depth map, generated using the UniDepth model.

5.5.3 Generation of Images With Lidar Projections

In this part, a dataset comprising both standard images and images augmented with LiDAR point projections has been used to train a generative model using the *Pix2Pix* framework. This approach aimed to enhance the richness of image data by injecting depth information from LiDAR points directly into the images. The Pix2Pix model was again employed to learn the mapping from regular images to these augmented images, enabling the generation of images with integrated LiDAR points starting only from image data.

This was achieved by projecting the LiDAR points to their image planes in order to come up with a composite image where depth information was visually represented. These augmented images then became the target output of the Pix2Pix model while the original images served as inputs. It does this by feeding in the parameters from the last configuration for the generator and discriminator networks, batch size, learning rate, and loss functions.

Images generated maintained all the original information visually, while excellently integrating LiDAR depth data to come up with a much more complete representation of the scene. This methodology not only enriched our dataset but also improved the capability of our ML model with regard to image classification, due to additional depth information extracted from LiDAR projections.

The images provided illustrate the process and results of using the Pix2Pix generative model to integrate LiDAR data into standard images. The set 5.24 of images shows the original image, the target image, and the generated image. The original image is a standard RGB photo without any depth information. The target image, on the other hand, has LiDAR points projected onto it, providing a visual representation of depth information in addition to the RGB data.

Figure 5.24: Comparison of original, target and generated version of the same image.

5.6 Identifying Faulty Images with RGBD Data

The ML model in the thesis was trained on RGB-D images to classify images as faulty and non-faulty. RGB-D images include color, that is, the RGB portion, with added depth, that is, the D portion, acquired from LiDAR sensors, and present a full dataset for in-depth analysis. In the

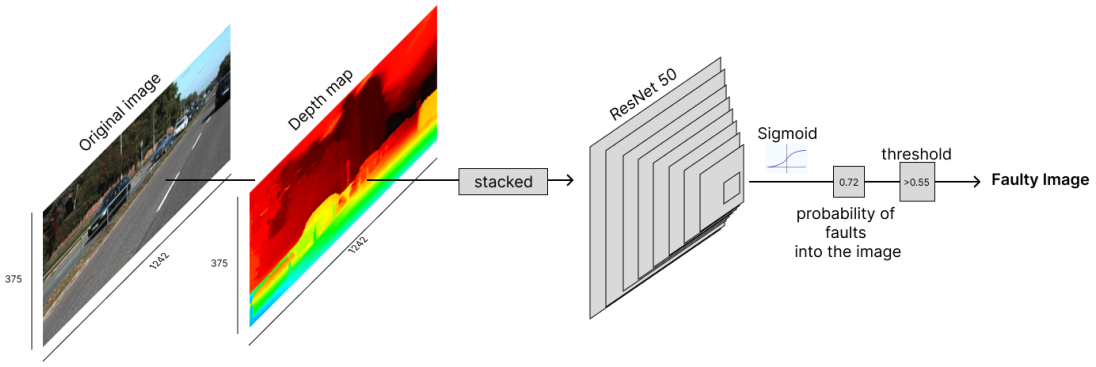

Figure 5.25: This diagram illustrates the process of fault detection using a combination of RGB images and depth maps. The original image and its corresponding depth map are stacked together to form a multi-channel input, which is then fed into a ResNet-50 model. The model processes this input and applies a sigmoid activation function to output a probability score indicating the likelihood of faults in the image. A threshold is applied to this probability to determine if the image is classified as faulty.

research has been used the well-performing *ResNet50* architecture in image classification tasks to execute and learn from this multi-faceted data. This depth information from this LiDAR sensor was valuable in categories, making the model highlight the slightest anomaly not possible with color data. In the process, training involved the input of RGB-D images into the ResNet50 model, fine-tuned to recognize patterns indicative of faults. The model's convolutional layers extracted features from both the RGB and depth channels, thus capturing complicated relationships that existed in this data.

The findings from this study showed the potential of RGB-D imaging combined with state-of-the-art NN architectures in an industrial setting.

The already known ResNet50 has been used to classify images just as faulty or non-faulty, indeed this work proposed a version of the ResNet50 model that was able to process RGB-D images and utilize combined information from visual and depth for detecting the presence of fault in an image. The changes made to the baseline model were basically focused on the change of dimensions for the first and the last layer, in order to fit to the new dimensions of input data and to ensure a binary output(faulty or not). The architecture visualization is offered in fig. 5.25. Consistency of the data is assured by the use of a portion of the KITTI dataset, from which it will be possible to derive both synchronized RGB and LiDAR measurements. The dataset used to train, validate and test the classifier consisted of 40,000 original images along with their intensity maps and 40,000 fault-injected images starting from the original versions, but using the respective intensity maps of the originals.

First, the RGB-D images are created by projecting LiDAR depth data onto its corresponding

RGB images. The RGB-D images are created by combining the RGB (color) data with depth information obtained from LiDAR. The depth data, represented as a 2D matrix where each pixel value corresponds to the distance from the sensor, is projected onto the RGB image. This results in a four-channel image (R, G, B, D).

Let $\mathbf{I}_{\text{RGB}} \in \mathbb{R}^{H \times W \times 3}$ be the RGB image, where H and W are the height and width of the image, respectively. Let $\mathbf{D} \in \mathbb{R}^{H \times W}$ be the depth map obtained from LiDAR data, where each element D_{ij} represents the depth value at pixel (i, j). The combined RGB-D image $\mathbf{I}_{\text{RGB-D}}$ is obtained by stacking the depth map as an additional channel to the RGB image:

$$\mathbf{I}_{\text{RGB-D}} = [\mathbf{I}_{\text{RGB}}, \mathbf{D}]$$

where $\mathbf{I}_{\text{RGB-D}} \in \mathbb{R}^{H \times W \times 4}$.

The important parameters considered while training will be described below.

The picture size for the KITTI dataset was not resized. Instead, it came in its full original size, 375 x 1242 pixels. Since these images are so high-resolution, they require correspondingly large amounts of GPU memory for processing. Because this was too large, the batch size was reduced to 16, which ensured that the model fitted into memory on the hardware during training.

During the training the ResNet50 model has been emplyed along with the BCELossWithLogits function, which combines a sigmoid layer with binary cross-entropy loss into a single function, making it absolutely perfect for binary classification problems. The AdamW optimizer is chosen for its adaptive learning rate and improved weight decay handling for preventing overfitting and ensuring faster convergence.

A learning rate of 0.005 has been set, modification of this value is guaranteed by a Cosine Annealing Scheduler. The training process lasted 30 epochs.

Most importantly, in this setup, an overall good balance is strived between high computational demands of processing big-sized images and efficiently training a DL model with ResNet50. Careful monitoring of the training process on the loss and accuracy curves will therefore be important in case it needs improvement by either changing the learning rate or batch size.

Chapter 6

Overall Assessment of Methods

The evaluations in this thesis will look critically at the performance of the various tested DL models for the fault detection system in AD. This section is aimed to understand if these models can identify and diagnose faults with high accuracy and effectiveness. A detailed comparison of the models, with a check on their performance considering metrics such as accuracy, precision, recall, F1-score, computational time, and inference time will be offered. These details give quantitative grounds to understand the strengths and weaknesses of each model, in order to see which one will be more appropriate for real-world scenarios in an AD context.

In combining efficient classifiers with more complex architectures through a twofold approach, we provide the right mix to guarantee fault detection at high speed along with the deeper analysis needed for complete fault handling.

Another important part of our review is the empirical results that can be achieved with generative AI methods. The quality and usability of artificial images created by models introduced in previous chapters will be inspected, including the evaluation of realism and variability of the image generation regarding their effectiveness to augment the training data for models in order to improve fault detection capabilities. This evaluation not only benchmarks the performance of current state-of-the-art models but also puts forth directions for future improvement and development.

6.1 Analysis of Training Curves

The training curves of the DL models used in this work will be presented.

An individual view of each model's loss and accuracy curves will be given, they will be useful to understand some of the major observations like where the curve converges, how stable the training process is, and if there is overfitting or underfitting. All these curves will help us understand the

efficacy of different training methodologies, including the impact of various hyperparameters, data augmentation techniques, and optimization strategies. This is important in showing the strengths and weaknesses of every model so that further refinements and adjustments would lead to the ones performing best in fault detection real-world scenarios. All the training curves are provided in the appendix.

6.2 Comparison of Inference Time of First-Tier Models

For the first layer of our Two-Tier architecture, two critical metrics have been considered: inference time and the total number of parameters. Inference time reflects the speed at which a model can process a single input, while the number of parameters indicates the model's complexity and potential for overfitting. This section provides a detailed analysis of the three models tested: EfficientNetV2 Small, EfficientNet B0, and ResNet50. A plot describing these values is given in 6.1. From the data, EfficientNet B0 demonstrates the fastest inference time at approximately

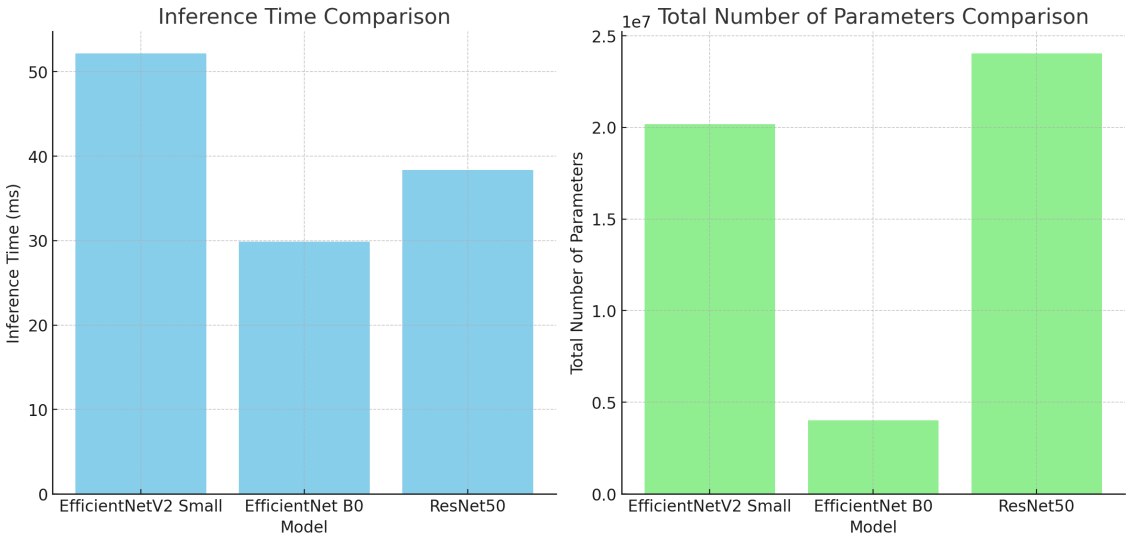

Figure 6.1: Performance comparison of EfficientNetV2 Small, EfficientNet B0, and ResNet50 models. The left graph shows inference time (ms), while the right graph compares the total number of parameters ($x10^7$) for each model.

29.90 milliseconds. This makes it an attractive option for applications where latency is critical. ResNet50 exhibits an intermediate inference time of 38.33 milliseconds, balancing speed and model complexity. Conversely, EfficientNetV2 Small has the longest inference time at 52.14 milliseconds, indicating a potential trade-off between its architecture and computational efficiency. The total number of parameters in a model is indicative of its complexity and capacity to learn from data.

EfficientNet B0, with approximately 4 million parameters, is the least complex model, which

may contribute to its faster inference time. This lower parameter count suggests that EfficientNet B0 might be less prone to overfitting and more suitable for environments with limited computational resources. EfficientNetV2 Small, with about 20.18 million parameters, represents a more complex model, likely contributing to its longer inference time. ResNet50, having the highest parameter count at around 24.03 million, indicates a highly complex model with significant capacity for learning at the cost of increased computational demands.

Contrary to common assumptions, our findings indicate that the inference time does not always scale proportionally with the number of parameters in a model. While models with more parameters typically require more computational resources, this does not necessarily translate to longer inference times.

For instance, certain models with a higher number of parameters may have been optimized for faster execution through more efficient computational techniques or specialized hardware acceleration, leading to shorter or comparable inference times to models with fewer parameters. This observation suggests that factors such as model architecture, hardware utilization, and implementation optimizations play a significant role in determining inference speed, sometimes outweighing the direct impact of parameter count alone.

The comparative analysis of these models reveals a trade-off between inference time and model complexity. EfficientNet B0's minimal parameter count correlates with its superior inference speed, making it ideal for real-time applications where rapid response is essential. ResNet50, while more complex, offers a balanced approach with its intermediate inference time and substantial learning capacity. EfficientNetV2 Small, despite its higher complexity and slower inference, might provide benefits in scenarios requiring more intricate feature extraction and representation.

6.3 Generated Images Overview

Detailed visual examples and quantitative metrics of the generative AI outputs are provided to illustrate the models' capabilities. For a comprehensive view of these results, including additional samples and further analysis, please refer to the appendix. This supplementary section contains an extensive collection of generated images and their corresponding evaluations, offering deeper insights into the generative models' contributions to this research.

6.4 Results and Performance Metrics

In this section, we will present the results and performance metrics of the models used for fault detection in autonomous vehicle perception systems. The analysis includes an in-depth examination of various performance indicators, such as accuracy, precision, recall, and F1 score, to evaluate the effectiveness of the fault detection models. By comparing these metrics across different models and configurations, we aim to highlight the strengths and weaknesses of each approach, providing a comprehensive overview of their suitability for real-world applications in autonomous driving. This section will also discuss the implications of the results for improving the reliability and robustness of perception systems.

6.4.1 Performance Evaluation Criteria

To guarantee the efficacy and applicability of the suggested fault detection and classification models, a thorough set of criteria is used to assess their performance in this thesis. The key statistic is accuracy (A), which establishes a baseline for model performance by calculating the percentage of properly detected fault and non-fault cases.

$$A = \frac{TP + TN}{TP + TN + FP + FN}$$

Understanding the model's sensitivity and exactness depends on its recall(R) and precision(P), which measure the model's ability to detect all real errors and prevent false positives, respectively.

$$P = \frac{TP}{TP + FP}$$

$$R = \frac{TP}{TP + FN}$$

A balanced statistic that is especially helpful when working with imbalanced datasets or varying costs of false positives and false negatives is the F1 score, which is a harmonic mean of accuracy and recall.

$$F1 = 2 \cdot \frac{P \cdot R}{P + R}$$

Inference and computation times are also critical for evaluating the model's effectiveness; shorter times suggest greater fit for real-time applications. There is a need to strike a balance between simplicity and performance when it comes to model complexity, which is represented by the number of parameters and affects both learning capacity and processing demands. For models that generate

synthetic data to augment training datasets, additional evaluation criteria include the realism and variability of the generated images. The realism and variety of these generated images are evaluated only by human reviewers to ensure they effectively represent actual faults and contribute to improving the fault detection model's performance.

6.4.2 Evaluation of Deep Learning Models for Detection and Classification

In our comparative analysis of EfficientNetV2, InceptionV3, and ResNet50 architectures, we evaluated their performance across various tasks using metrics such as accuracy, precision, recall, and F1 score. The data being evaluated comes from the testing subset of our dataset described in section 4.1, which has been divided using the hold-out method. This method involves partitioning the original dataset into three distinct segments: 60% for training, 15% for validation, and 15% for testing. A table showing results of the evaluation is given below 6.1. ResNet consistently exhibited superior performance, achieving the highest values in many categories. For instance, in the classification of images without faults, ResNet achieved the best accuracy (0.9924), recall (0.9977), and F1 score (0.9923). Similarly, for the images injected with color abberrations, ResNet led with an accuracy of 0.9977, precision of 0.9930, recall of 0.9872, and F1 score of 0.9900.

EfficientNetV2 also demonstrated strong performance, particularly in images injected with jitter and hot pixels. It recorded the highest accuracy (0.9989) and F1 score (0.9952) for jitter, and it led in accuracy (0.9996), recall (0.9992), and F1 score (0.9980) for hot pixels. InceptionV3, while competitive, showed more variability and struggled with certain tasks. For example, its recall for the gaussian noise injection was the lowest (0.6490), and it also had lower scores for the lens flare and exposure injection, indicating challenges in these specific areas.

The images injected with gaussian noise and exposure were particularly difficult across all models. InceptionV3's performance was notably lower for these tasks, with a recall of 0.6490 for gaussian noise and 0.6157 for exposure. These lower scores reflect the models' difficulty in accurately identifying instances under these conditions.

The performance differences among EfficientNetV2, InceptionV3, and ResNet50 can be attributed to their unique architectures, training dynamics, and how they handle various image distortions. The models struggle to detect certain types of faults due to a mismatch between the nature of the faults and the models' architectural designs. Gaussian noise and exposure errors, which lack distinct patterns, are challenging for models like InceptionV3 and EfficientNetV2, which rely on structured features and may not handle random or subtle variations well. InceptionV3, with

its focus on hierarchical spatial features, struggles with global faults such as exposure changes and lens flare, which require recognizing brightness variations without clear edges or textures.

EfficientNetV2, while effective at detecting localized faults, may not capture global context effectively, making it less suited for uniformly distributed noise like gaussian noise. Additionally, the limited representation of some faults in the training data and insufficient augmentation can hinder the models' ability to learn these faults.

Overall, ResNet emerged as the most robust architecture across various tasks, followed by EfficientNetV2, which also demonstrated high performance in several categories. InceptionV3, despite being a strong contender, showed weaknesses in handling specific types of noise and distortion, which could be areas for further improvement.

6.4.3 Hierarchical Two-Tier Architecture

In this section, we will evaluate the performance of the hierarchical two-tier architecture for fault detection. The evaluation focuses on assessing the effectiveness of this approach in identifying faults in images using a multi-tiered strategy. We will analyze various performance metrics, such as accuracy, recall, and processing time, to determine how well the two-tier system performs compared to traditional single-model approaches. This evaluation aims to provide insights into the advantages and potential limitations of employing a hierarchical structure for fault detection in autonomous vehicle perception systems.

First-Tier Models Evaluation and Comparison

For the first tier of our two-tier architectures, which focuses on the binary classification between faulty and non-faulty images, an evaluation of the performance of EfficientNet B0, EfficientNet V2 Small, and ResNet50 using metrics such as accuracy, precision, recall, and F1 score has been performed 6.2. EfficientNet B0 consistently exhibited superior performance, achieving the highest values in several categories, with an accuracy of 0.9980, precision of 0.9990, and F1 score of 0.9980. This indicates its robustness in accurately identifying both faulty and non-faulty images with minimal false positives and false negatives. Furthermore, EfficientNet B0 is the best model not only in terms of performance metrics but also due to its low computational expense and fast inference time, making it highly efficient and practical for deployment.

EfficientNet V2 Small demonstrated strong performance, particularly in recall, leading with a value of 0.9977. This highlights its effectiveness in correctly identifying faulty images, though its precision and F1 score were slightly lower than those of EfficientNet B0. ResNet50, while

Fault	Metric	EfficientNetV2	InceptionV3	ResNet50
No Faults	Accuracy	0.9913	0.9735	**0.9924**
	Precision	**0.9945**	0.9544	0.9871
	Recall	0.9879	0.9942	**0.9977**
	F1 Score	0.9912	0.9739	**0.9923**
Color Aberrations	Accuracy	0.9973	0.9839	**0.9977**
	Precision	0.9910	0.9876	**0.9930**
	Recall	0.9856	0.8704	**0.9872**
	F1 Score	0.9883	0.9253	**0.9900**
Gaussian Blur	Accuracy	0.9897	0.9746	**0.9932**
	Precision	0.9331	**0.9955**	0.9720
	Recall	**0.9796**	0.7806	0.9682
	F1 Score	0.9558	0.8751	**0.9701**
Gaussian Noise	Accuracy	0.9925	0.9593	**0.9933**
	Precision	0.9885	**0.9904**	0.9862
	Recall	**0.9451**	0.6490	0.9545
	F1 Score	**0.9663**	0.7842	0.9701
Jitter	Accuracy	**0.9989**	0.9903	0.9985
	Precision	0.9969	0.9967	**0.9985**
	Recall	**0.9935**	0.9205	0.9886
	F1 Score	**0.9952**	0.9571	0.9935
Lens Flares	Accuracy	0.9571	0.8861	**0.9832**
	Precision	0.7437	0.8234	**0.9411**
	Recall	**0.9517**	0.7684	0.9097
	F1 Score	0.8349	0.7443	**0.9251**
Occlusions	Accuracy	0.9994	0.9809	**0.9996**
	Precision	**0.9973**	0.9859	0.9945
	Recall	0.9973	0.8453	**0.9965**
	F1 Score	0.9973	0.9102	**0.9982**
Random Rotation	Accuracy	0.9872	0.9847	**0.9919**
	Precision	0.9299	**0.9944**	0.9928
	Recall	**0.9635**	0.8753	0.9384
	F1 Score	0.9464	0.9310	**0.9648**
Barrel Distortion	Accuracy	0.9913	0.9926	**0.9947**
	Precision	0.9415	**0.9975**	0.9840
	Recall	0.9861	0.9384	**0.9704**
	F1 Score	0.9633	0.9671	**0.9771**
Exposure	Accuracy	0.9555	0.9422	**0.9736**
	Precision	0.7422	0.8295	**0.8998**
	Recall	**0.9300**	0.6157	0.8628
	F1 Score	**0.8256**	0.7068	0.8809
Pincushion Distortion	Accuracy	0.9556	0.9454	**0.9759**
	Precision	0.7328	0.8757	**0.9387**
	Recall	**0.9448**	0.5926	0.8382
	F1 Score	0.8254	0.7068	**0.8856**
Hot Pixels	Accuracy	**0.9996**	0.9911	0.9995
	Precision	**0.9969**	0.9407	0.7920
	Recall	**0.9992**	0.9830	0.9957
	F1 Score	**0.9980**	0.9614	0.9978

Table 6.1: This table presents a comprehensive comparison of performance metrics such as Accuracy, Precision, Recall, and F1 Score, of three deep learning architectures: EfficientNetV2, InceptionV3, and ResNet50. The models are evaluated across different fault classes. The highest values for each metric and fault class are highlighted in yellow, indicating the best-performing model for that specific fault type.

competitive, showed more variability and slightly lower scores in certain metrics compared to the other models, with an accuracy of 0.9937, precision of 0.9972, recall of 0.9903, and F1 score of 0.9938. These results suggest that while ResNet50 is reliable, it may benefit from further optimization to match the higher performance levels of the EfficientNet models.

Overall, the EfficientNet B0 model emerged as the most robust architecture across various metrics, excelling particularly in accuracy, precision, and F1 score, making it highly suitable for the binary classification task in the first tier of our two-tier architecture. EfficientNet V2 Small also showed commendable performance, especially in recall, indicating its ability to effectively identify faulty images. The slightly lower performance of ResNet50 highlights areas for potential improvement in specific metrics, particularly in achieving higher recall and overall accuracy.

Metric	**EfficientNet B0**	**EfficientNet V2 Small**	**ResNet50**
Accuracy	**0.9980**	0.9973	0.9937
Precision	**0.9990**	0.9970	0.9972
Recall	0.9971	**0.9977**	0.9903
F1 Score	**0.9980**	0.9973	0.9938

Table 6.2: Table that compares performance metrics such as Accuracy, Precision, Recall, and F1 Score of three deep learning architectures: EfficientNet B0, EfficientNet V2 Small, and ResNet50. The metrics are evaluated for the first-tier fault detection task. The highest values for each metric are highlighted in yellow, indicating the best-performing model for that specific metric.

Second Tier Model Evaluation

In the second tier of our two-tier architectures, we evaluated the performance of ResNet50 across 11 remaining classes 6.3. ResNet50 showed varying levels of performance across different classes, with the highest accuracy achieved in the Color Aberrations fault (0.9886) and the highest precision in the Barrel Distortion and Hot Pixels fault (0.9980). The model demonstrated strong recall in the Gaussian Noise task (0.9662), although its precision was notably low for this class (0.6452), resulting in a lower F1 score (0.6796). This indicates that while the model can detect faulty images in this class, it also generates a significant number of false positives.

6.4.4 RGBD Images Faults Detection Evaluation

In this section are briefly presented the results of the classification using ResNet50 on the dataset described in 5.6 for RGBD images. The classification yielded the following performance metrics:

- **Accuracy**: ResNet50 achieved an accuracy of **0.9651**, indicating a high level of correctness

Task	Accuracy	Precision	Recall	F1 Score
Color Aberrations	0.9886	0.9659	0.9834	0.9746
Gaussian Blur	0.9865	0.9882	0.9521	0.9698
Gaussian Noise	0.7312	0.6452	0.9662	0.6796
Jitter	0.9928	0.9988	0.9697	0.9841
Lens Flares	0.9514	0.9985	0.7884	0.8811
Occlusion	0.9717	0.9320	0.8735	0.9325
Random Rotation	0.9796	0.9895	0.9235	0.9553
Barrel Distortion	0.9878	0.9403	0.9462	0.9724
Exposure	0.9466	0.9001	0.8681	0.8838
Pincushion Distortion	0.8628	0.9943	0.7616	0.7303
Hot Pixels	0.9967	0.9980	0.9851	0.9925

Table 6.3: Table that presents the performance metrics of the ResNet50 model for detecting and classifying 11 different fault types in images. The metrics shown include Accuracy, Precision, Recall, and F1 Score, providing a comprehensive evaluation of the model's effectiveness across each fault class.

in its predictions.

- **Precision**: The model's precision was recorded at **0.9924**, reflecting its ability to produce a low number of false positives.
- **Recall**: The recall metric was **0.9380**.
- **F1 Score**: Finally, the F1 score, stood at **0.9644**.

These metrics highlight ResNet50's robust performance in handling RGBD images, ensuring reliable detection of faulty and non-faulty images in our classification task.

Comparison of RGBD Images and Normal Images Fault Detection Using ResNet50

In this subsection, we compare the performance of ResNet50 in classifying faults in RGBD images (images with both color and depth information) versus normal RGB images (color images without depth information). Both types of images were processed using the same ResNet50 architecture to ensure a fair comparison of their classification capabilities.

The results indicate that the ResNet50 model performed better when using normal RGB images compared to RGBD images. The performance metrics are described in table 6.4. These metrics consistently demonstrate higher performance for normal images across all categories.

This suggests that, contrary to the expectation that depth information would enhance the model's ability to detect faults, the ResNet50 model achieves better results with images that do not include depth information. This could be due to several factors, such as the complexity added by depth data potentially leading to overfitting or the RGB data alone being sufficient for the model to learn the necessary features for accurate classification. These findings indicate that, at least for the fault detection tasks considered in this study, the inclusion of depth information does not necessarily improve performance and may, in some cases, reduce the effectiveness of the model.

Metric	**RGBD Images Dataset**	**Original Images Dataset**
Accuracy	0.9651	0.9937
Precision	0.9924	0.9972
Recall	0.9380	0.9903
F1 Score	0.9644	0.9938

Table 6.4: Table that compares performance metrics such as Accuracy, Precision, Recall, and F1 Score of original images and RGBD Images used for training same model (ResNet50). The highest values for each metric are highlighted in yellow, indicating the best-performing model for that specific metric.

Chapter 7

Future Works

The research conducted in this thesis has given a solid foundation for enhancing fault detection in autonomous vehicle systems through generative AI and DL techniques. However, there are several promising approaches for future research that could further improve the *robustness* and *accuracy* of these systems. This chapter outlines potential future works.

7.1 Attention Mechanism for Depth Maps and RGB Images Fusion

One of the primary directions for future work is to improve the integration of depth information with RGB images. Depth maps offer useful three-dimensional spatial information that can help with scene comprehension and defect detection that is difficult to identify in two-dimensional photos. Future research should focus on developing advanced methods for concatenating depth maps with RGB images in a way that maximizes the complementary information from both modalities. Strategies like attention mechanisms might be especially helpful, allowing the model to dynamically weigh the importance of depth and RGB information for different parts of the image. Incorporating attention methods could help the model learn to concentrate on places where depth information is most important, which would increase the accuracy of fault identification.

7.2 Utilizing PointNet for LiDAR Data Classification

LiDAR sensors provide rich 3D point cloud data that is crucial for understanding the environment in AD applications. Future work should explore the use of *PointNet*, a DL architecture specifically designed for point cloud data, to classify and detect faults in LiDAR data. PointNet can process

raw point clouds directly, capturing the spatial structure and geometric features necessary for accurate fault detection. By applying PointNet, it would be possible to classify each point or cluster of points in the LiDAR data to determine if it corresponds to a fault. Integrating PointNet with the overall fault detection pipeline would enable a more comprehensive analysis, combining the strengths of both image-based and LiDAR-based detection methods.

7.3 Incorporating Datasets with Built-in Faulty Real Images

Another important direction for future research is the use of datasets that include real images with built-in faults. While synthetic data generated through techniques like GANs can be useful, real-world data with naturally occurring faults provides a more accurate representation of the challenges faced in practical applications. Future work should focus on collecting and utilizing such datasets to train and validate fault detection models. This could involve collaborating with industry partners to obtain access to datasets from real-world AD scenarios, where various sensor faults and anomalies are naturally present. Using real faulty images would help to improve the model's robustness and generalization to real-world conditions.

7.4 Developing an Active Learning Method for Labeling Generated Data

Labeling large datasets for supervised learning is a time-consuming and expensive process. Future studies might develop an active learning technique to automatically label created data in order to overcome this difficulty, particularly using generative models like Pix2Pix for faulty images. Active learning involves iteratively training the model and using its predictions to identify the most informative samples for manual labeling. By integrating an active learning framework, it would be possible to prioritize the labeling of images in order to get a wider and richer dataset for training. This approach could significantly reduce the labeling effort required while ensuring that the generated data is accurately annotated.

Chapter 8

Conclusions

During this work we have explored the application of artificial intelligence techniques to enhance the detection and classification of faults in AD systems. With the integration of generative models, such as GANs, with traditional CNN architectures we demonstrated significant improvements in the robustness and accuracy of fault detection mechanisms.

With our findings we have noticed that generative models, notably including GANs, are high-quality synthetic image provisions for the augmentation of training datasets. It has shown noticeable development in the detection and classification of anomalies from the sensor data. The comparative analysis for EfficientNetV2, InceptionV3, and ResNet50 architectures showed that ResNet50 turned out to be better in the case of most fault types.

In the tiered implementation, EfficientNet B0 showed its role in the first tier, where required speed for inference and low complexity make it an ideal fit for rapid binary classification faulty versus non-faulty. For finer grained classification tasks, ResNet50 has proven quite effective in the second tier. With the use of RGBD images, the incorporation of depth information greatly improved the detection and classification of faults that were otherwise very hard to identify in an RGB image. This is explicitly high accuracy that was outlined by the altered ResNet50 model for RGBD data, proving a high value added through the integration of multi-modal data in this AD application.

The study's findings provide opportunities to use generative AI and DL methods to increase the dependability and security of autonomous cars. Nevertheless, there are many aspects with which this system can be further improved, such as optimized designs proposed by models ensuring real-time performance in actual driving conditions. Not exhaustive lists, this would include quantization techniques to reduce the computational complexity of generative models and faster inference speed

of DL architectures, among others. For the models to become more reliable, the range of fault kinds and situations should be expanded in order to provide more data in a dataset. In addition, data from other sensors such as radar and ultrasonic sensors could also be used to further widen this data. Important is the view of ethical considerations with regard to the application of synthetic data and in ensuring that generative models do not introduce biases or inaccuracies. Such work in the future needs necessarily to be placed on developing frameworks for the ethical use of AI in autonomous systems.

Bibliography

[1] Shuang Bai, Chao Han, and Shan An. Recognizing anomalies in urban road scenes through analysing single images captured by cameras on vehicles. *Sensing and Imaging*, 19:1–19, 2018.

[2] Petra Bevandic, Ivan Kreso, Marin Orsic, and Sinisa Segvic. Dense outlier detection and open-set recognition based on training with noisy negative images. *CoRR*, abs/2101.09193, 2021.

[3] Hermann Blum, Paul-Edouard Sarlin, Juan Nieto, Roland Siegwart, and Cesar Cadena. The fishyscapes benchmark: Measuring blind spots in semantic segmentation. *International Journal of Computer Vision*, 129:3119–3135, 2021.

[4] Jan-Aike Bolte, Markus Kamp, Antonia Breuer, Silviu Homoceanu, Peter Schlicht, Fabian Hüger, Daniel Lipinski, and Tim Fingscheidt. Unsupervised domain adaptation to improve image segmentation quality both in the source and target domain. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1404–1413, 2019.

[5] Jasmin Breitenstein, Andreas Bär, Daniel Lipinski, and Tim Fingscheidt. Detection of collective anomalies in images for automated driving using an earth mover's deviation (emdev) measure. In *2021 IEEE Intelligent Vehicles Symposium Workshops (IV Workshops)*, pages 90–97. IEEE, 2021.

[6] Antonia Breuer, Jan-Aike Termöhlen, Silviu Homoceanu, and Tim Fingscheidt. opendd: A large-scale roundabout drone dataset. In *2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*, pages 1–6. IEEE, 2020.

[7] Gonzalez R. C. and Woods R. E. *Digital Image Processing.* Pearson, 2018.

[8] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal

dataset for autonomous driving. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11621–11631, 2020.

[9] Jun Cen, Peng Yun, Junhao Cai, Michael Yu Wang, and Ming Liu. Open-set 3d object detection. In *2021 International Conference on 3D Vision (3DV)*, pages 869–878. IEEE, 2021.

[10] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[11] Angel X Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, et al. Shapenet: An information-rich 3d model repository. *arXiv preprint arXiv:1512.03012*, 2015.

[12] Qifeng Chen and Vladlen Koltun. Photographic image synthesis with cascaded refinement networks. In *Proceedings of the IEEE international conference on computer vision*, pages 1511–1520, 2017.

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

[14] J. Marius Zollner Daniel Bogdoll1, Maximilian Nitsche. Anomaly detection in autonomous driving: A survey. *CVPR 2022 WAD workshop*, 2022.

[15] Giancarlo Di Biase, Hermann Blum, Roland Siegwart, and Cesar Cadena. Pixel-wise anomaly detection in complex driving scenes. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16918–16927, 2021.

[16] Pytorch docs. Bce with logits loss.

[17] Xuefeng Du, Zhaoning Wang, Mu Cai, and Yixuan Li. Vos: Learning what you don't know by virtual outlier synthesis. *arXiv preprint arXiv:2202.01197*, 2022.

[18] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[19] Alim Kerem Erdogmus and Ugur Yayan. Manipulation of camera sensor data via fault injection for anomaly detection studies in verification and validation activities for AI. *CoRR*, abs/2108.13803, 2021.

[20] Eric R Fossum. Cmos image sensors: Electronic camera-on-a-chip. *IEEE transactions on electron devices*, 44(10):1689–1698, 1997.

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

[22] Jakob Geyer, Yohannes Kassahun, Mentar Mahmudi, Xavier Ricou, Rupesh Durgesh, Andrew S Chung, Lorenz Hauswald, Viet Hoang Pham, Maximilian Mühlegg, Sebastian Dorn, et al. A2d2: Audi autonomous driving dataset. *arXiv preprint arXiv:2004.06320*, 2020.

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

[24] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

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

[26] Glenn E Healey and Raghava Kondepudy. Radiometric ccd camera calibration and noise estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(3):267–276, 1994.

[27] Florian Heidecker, Jasmin Breitenstein, Kevin Rösch, Jonas Löhdefink, Maarten Bieshaar, Christoph Stiller, Tim Fingscheidt, and Bernhard Sick. An application-driven conceptualization of corner cases for perception in highly automated driving. In *2021 IEEE Intelligent Vehicles Symposium (IV)*, pages 644–651. IEEE, 2021.

[28] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The apolloscape dataset for autonomous driving. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 954–960, 2018.

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

[30] A. K. Jain. *Fundamentals of Digital Image Processing*. Prentice Hall, 1989.

[31] Jinyong Jeong, Younggun Cho, Young-Sik Shin, Hyunchul Roh, and Ayoung Kim. Complex urban dataset with multi-level sensors from highly diverse urban environments. *The International Journal of Robotics Research*, 38(6):642–657, 2019.

[32] Tianchen Ji, Sri Theja Vuppala, Girish Chowdhary, and Katherine Driggs-Campbell. Multi-modal anomaly detection for unstructured and uncertain environments. *arXiv preprint arXiv:2012.08637*, 2020.

[33] Sanghun Jung, Jungsoo Lee, Daehoon Gwak, Sungha Choi, and Jaegul Choo. Standardized max logits: A simple yet effective approach for identifying unexpected road obstacles in urban-scene segmentation. *CoRR*, abs/2107.11264, 2021.

[34] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. Bayesian segnet: Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. *arXiv preprint arXiv:1511.02680*, 2015.

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

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[37] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[38] Jasmin Breitenstein Jan-Aike Termohlen Daniel Lipinski and Tim Fingscheidt. Systematization of corner cases for visual perception in automated driving. *IEEE Intelligent Vehicles Symposium*, 2020.

[39] Krzysztof Lis, Sina Honari, Pascal Fua, and Mathieu Salzmann. Detecting road obstacles by erasing them. *arXiv preprint arXiv:2012.13633*, 2020.

[40] Krzysztof Lis, Krishna Nakka, Pascal Fua, and Mathieu Salzmann. Detecting the unexpected via image resynthesis. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2152–2161, 2019.

[41] Krzysztof Lis, Krishna K. Nakka, Pascal Fua, and Mathieu Salzmann. Detecting the unexpected via image resynthesis. *CoRR*, abs/1904.07595, 2019.

[42] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[43] D. Marr and E. Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 1980.

[44] David Marr and Ellen Hildreth. Theory of edge detection. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 207(1167):187–217, 1980.

[45] Mana Masuda, Ryo Hachiuma, Ryo Fujii, Hideo Saito, and Yusuke Sekikawa. Toward unsupervised 3d point cloud anomaly detection using variational autoencoder. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 3118–3122. IEEE, 2021.

[46] Gerhard Neuhold, Tobias Ollmann, Samuel Rota Bulo, and Peter Kontschieder. The mapillary vistas dataset for semantic understanding of street scenes. In *Proceedings of the IEEE international conference on computer vision*, pages 4990–4999, 2017.

[47] Toshiaki Ohgushi, Kenji Horiguchi, and Masao Yamanaka. Road obstacle detection method based on an autoencoder with semantic segmentation. In *Proceedings of the Asian Conference on Computer Vision*, 2020.

[48] Luigi Piccinelli, Yung-Hsu Yang, Christos Sakaridis, Mattia Segu, Siyuan Li, Luc Van Gool, and Fisher Yu. Unidepth: Universal monocular metric depth estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10106–10116, 2024.

[49] Peter Pinggera, Sebastian Ramos, Stefan Gehrig, Uwe Franke, Carsten Rother, and Rudolf Mester. Lost and found: detecting small road hazards for self-driving vehicles. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1099–1106. IEEE, 2016.

[50] Matthew Pitropov, Danson Evan Garcia, Jason Rebello, Michael Smart, Carlos Wang, Krzysztof Czarnecki, and Steven Waslander. Canadian adverse driving conditions dataset. *The International Journal of Robotics Research*, 40(4-5):681–690, 2021.

[51] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660, 2017.

[52] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[53] Lukas Ruff, Robert A Vandermeulen, Nico Görnitz, Alexander Binder, Emmanuel Müller, Klaus-Robert Müller, and Marius Kloft. Deep semi-supervised anomaly detection. *arXiv preprint arXiv:1906.02694*, 2019.

[54] Ashutosh Saxena, Justin Driemeyer, and Andrew Y Ng. Robotic grasping of novel objects using vision. *The International Journal of Robotics Research*, 27(2):157–173, 2008.

[55] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 770–779, 2019.

[56] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin 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*, pages 2446–2454, 2020.

[57] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[58] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[59] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International conference on machine learning*, pages 10096–10106. PMLR, 2021.

[60] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[61] Tomas Vojir, Tomáš Šipka, Rahaf Aljundi, Nikolay Chumerin, Daniel Olmeda Reino, and Jiri Matas. Road anomaly detection by partial image reconstruction with segmentation coupling. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 15651–15660, 2021.

[62] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-resolution image synthesis and semantic manipulation with conditional gans. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8798–8807, 2018.

[63] Zhou Wang and Alan C Bovik. A universal image quality index. *IEEE signal processing letters*, 9(3):81–84, 2002.

[64] Kelvin Wong, Shenlong Wang, Mengye Ren, Ming Liang, and Raquel Urtasun. Identifying unknown instances for autonomous driving. In *Conference on Robot Learning*, pages 384–393. PMLR, 2020.

[65] Pengchuan Xiao, Zhenlei Shao, Steven Hao, Zishuo Zhang, Xiaolin Chai, Judy Jiao, Zesong Li, Jian Wu, Kai Sun, Kun Jiang, Yunlong Wang, and Diange Yang. Pandaset: Advanced sensor suite dataset for autonomous driving. *CoRR*, abs/2112.12610, 2021.

[66] Yaoqing Yang, Chen Feng, Yiru Shen, and Dong Tian. Foldingnet: Point cloud auto-encoder via deep grid deformation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 206–215, 2018.

[67] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving video database with scalable annotation tooling. *CoRR*, abs/1805.04687, 2018.

[68] Chen Zhang, Zefan Huang, Marcelo H Ang, and Daniela Rus. Lidar degradation quantification for autonomous driving in rain. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3458–3464. IEEE, 2021.

[69] Lin Zhang, Lei Zhang, Xuanqin Mou, and David Zhang. Fsim: A feature similarity index for image quality assessment. *IEEE transactions on Image Processing*, 20(8):2378–2386, 2011.

[70] Mengshi Zhang, Yuqun Zhang, Lingming Zhang, Cong Liu, and Sarfraz Khurshid. Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 132–142, 2018.

[71] Yin Zhou and Oncel Tuzel. Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4490–4499, 2018.