https://doi.org/10.2352/EI.2026.38.16.AVM-104

CIDPL: A real-time CUDA-accelerated Python-framework simulating PSF-based optical artifacts integrated in MMDetection

Maximilian Dornik [1]***, Julian Barthel*** [1]***, Daniel Jakab*** [2] ***and Alexander Braun*** [1]
[1] ***Faculty of Electrical Engineering & Information Technology, University of Applied Sciences Düsseldorf, Germany***
[2] ***Dept. of Electronic and Computer Engineering, University of Limerick, Castletroy, Co. Limerick V94 T9PX, Ireland***

Abstract

Physically grounded PSF-based image degradation is necessary for studying optics-related object-detector robustness, but existing workflows typically rely on offline dataset generation and integrate poorly with GPU-resident frameworks such as MMDetection. We present CIDPL, a CUDA-accelerated Python library that adapts our standalone Image Degradation Application (IDA) into a real-time, framework-integrated pipeline for MMDetection. CIDPL couples Python and C++ via PyBind11, performs degradation directly on GPU tensors in the DataPreprocessor, and organizes multiple optical variants in a traceable Super-Batch format. Numerical validation shows exact agreement with IDA for TIFF inputs, while optical validation reproduces KrakenOS-based SFR trends. In throughput tests, CIDPL improves mean degradation speed over IDA by 4.7x on a single GPU and 7.6x on two GPUs, enabling real-time processing at 117 FPS with negligible overhead during both training and inference. KITTI experiments further show that the integration enables practical detector-level robustness studies under varying defocus conditions.

Introduction

Object detection for 2D camera imagery is a key component of modern perception systems, especially in autonomous driving. In such safety-critical settings, robustness against real optical conditions is essential. While optics-induced domain shift is often under-modeled in current object detection workflows, missed detections can directly lead to safety-relevant failures. [23, 24, 19]

Real optical systems, e.g. automotive lenses, are shaped by physically grounded effects such as field-dependent defocus and aberrations, which are naturally described by Point Spread Functions (PSFs). Thus, robustness to optics-induced artifacts must be assessed under PSF-based degradations tied to measurable properties of concrete lens designs, rather than relying only on simplified image perturbations. [34, 25, 13] Current object detection workflows typically rely on one of two extremes.

Simplified augmentations such as Gaussian blur are easily integrated, but do not represent realistic optical image formation [25]. Conversely, physically motivated PSF-based degradation is often realized offline, e.g. through our self-developed, CUDA-accelerated Image Degradation Application (IDA), by pre-generating degraded dataset variants [4]. This improves realism but incurs storage overhead, management complexity and limited runtime flexibility. This gap is particularly problematic for robustness studies, requiring many optical conditions per image and explicit attribution of predictions to the applied degradation variant.

We address this missing middle ground by integrating physically grounded PSF-based image degradation through our novel CUDA Image Degradation Python Library (CIDPL), derived from IDA, into the widely used MMDetection framework [8] for object detection. This enables a unified and efficient platform for training and evaluating detectors under realistic optical degradations. Since IDA is our implementation and already serves as a fundament for SOLAS by Jakab et al. [18, 17], it is the natural implementation reference for this work. Although demonstrated in MMDetection, CIDPL is designed to be framework-agnostic and can be reused in arbitrary PyTorch-based pipelines beyond object detection. Fig. 1 illustrates the core idea.

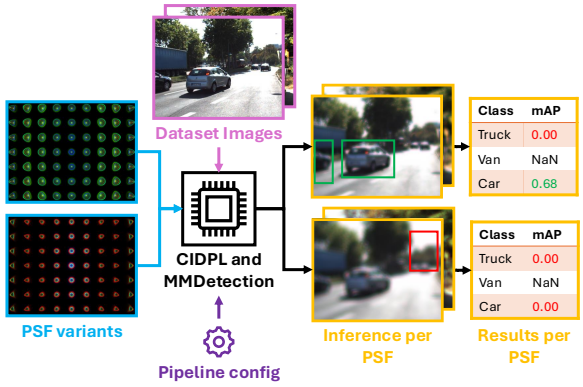


Figure 1: *Overview of the proposed CIDPL-based MMDetection pipeline.*

Dataset images are combined with multiple PSF variants within a unified pipeline to run training and inference under different optical conditions and to obtain performance measures per variant without detouring through offline dataset generation. Importantly, the novelty of this work is not the general ability to degrade images realistically on a GPU, but the execution model that makes physically grounded image degradation operationally viable in large-scale object detection. Effective integration of IDA-style degradation into MMDetection is not straightforward.

MMDetection expects batched tensor processing inside a Python-based, GPU-resident pipeline [22, 5], whereas IDA was originally designed as an offline, file-oriented C++-application that processes single images at runtime [4]. This mismatch motivates the central novelties of the paper: a direct Python-to-C++ coupling via PyBind11, GPU-resident degradation at the DataPreprocessor stage to avoid unnecessary CPU-GPU-transfers, and a structured Super-Batch representation that enables batch-parallel, multi-variant Test-Time Augmentation (TTA)-analogous [20] traceable evaluation over many PSF variants while providing consistent data flow. In addition, the integration exposes opti-

mization potential beyond the original IDA, whose single-image execution model is extended toward scalable, high-throughput, real-time-capable processing. Although the resulting toolchain can also be used for training, the integration problem addressed here is centered around evaluation and inference. In this setting, efficient execution, structured processing of many PSF variants, and per-variant traceability are the main practical requirements. To validate the proposed approach, we follow a three-part evaluation strategy considering numerical consistency, physical plausibility and practical utility. In summary, this paper makes the following contributions:

1. **CIDPL library:** A Python-accessible, CUDA-accelerated optical degradation library derived from IDA, reusable within arbitrary PyTorch workflows.
2. **Integration strategy:** A method for integrating high-throughput PSF-based degradation into MMDetection through direct Python-C++-coupling and GPU-resident tensor processing.
3. **Super-Batch structure:** A traceable representation for batch-parallel processing of multiple PSF variants.
4. **Validation:** Numerical agreement with IDA, optical plausibility against KrakenOS [15] Spatial Frequency Response (SFR) ground truth, effective throughput analysis and detection-level utility demonstration on KITTI [12] data.

Related Work

Robustness to image degradations is commonly studied using standardized corruption benchmarks such as ImageNet-C [14] and their extensions to detection tasks, including COCO-C, Pascal-C, and Cityscapes-C [21]. More recently, optics-oriented benchmarks such as OpticsBench and LensCorruptions have emphasized the need for lens-realistic degradations beyond generic synthetic corruptions [24]. In imaging science, physically meaningful blur is often modeled by PSFs derived from optical simulation or measurement [34]. Ray-tracing based pipelines such as SOLAS [18, 17], together with optical simulation tooling and standardized sharpness assessment via ISO 12233 [16], further highlight the importance of physically grounded image degradation. However, these approaches are frequently realized as separate simulation or offline processing workflows rather than being integrated directly into deep learning pipelines, which complicates their use in large-scale robustness studies.

Theoretical background

In this chapter, the underlying optical degradation model, MMDetection data-path assumptions, and IDA's reference offline execution model are described, establishing the technical system properties from which the proposed integration design follows.

The optical degradation model

Real camera lenses exhibit field-dependent optical behavior. Blur characteristics and artifact patterns change across the sensor, typically becoming more pronounced towards the image periphery. [19, 23] IDA supports space-variant degradation by representing a lens through a PSF-grid of shape $(m \times n)$, i.e. a set of locally valid, but spatially interpolated kernel patches with a defined pixel resolution, distributed over the image field (Fig. 2) [4].

These kernels are applied to the image through a superposition approach, whereby each pixel is influenced by a weighted

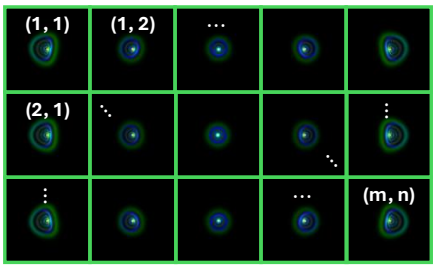


Figure 2: *Example of a 3×5 PSF-grid with 150 px resolution per side [1].*

combination of nearby PSF-kernels [4, 33]. In addition, IDA (and consequently CIDPL) can operate with channel-specific RGB-PSF-kernels [4]. This enables the simulation of chromatic effects such as color fringing induced by chromatic aberrations, alongside other position- and wavelength-dependent artifacts that cannot be captured by simplified blur augmentations.

The regular data path of MMDetection

MMDetection provides a modular pipeline for object detection training and evaluation. In typical configurations, data is processed by a sequence of framework components orchestrated by a Loop, managed within a Runner (Fig. 3). [28]

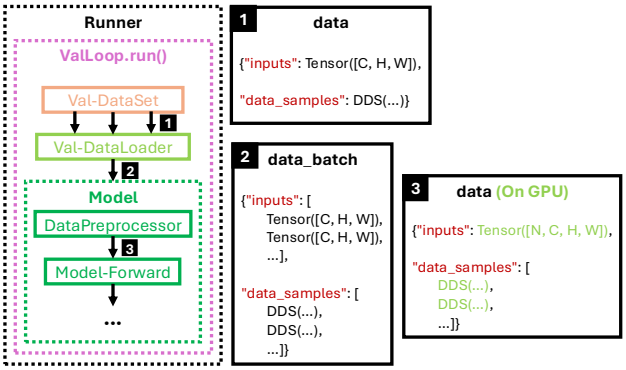


Figure 3: *Schematic of the MMDetection data flow during evaluation [1].*

The DataSet provides sample dictionaries containing image data together with annotations and metadata. After collation by the DataLoader, the DataPreprocessor converts the batch into model-ready GPU-memory tensors and associated DetDataSample (DDS) annotation containers. [1, 28, 11, 2, 5] For the present work, two properties of this workflow are particularly relevant.

First, MMDetection relies on a defined batch interface between data loading, preprocessing, model inference, and evaluation. Image content is represented as tensors with shape (N,C,H,W), where N denotes batch size, C the channel dimension, and H and W the image's height and width respectively. [1, 10, 2, 22] Any additional processing step must therefore remain compatible with this tensor-based batch representation and with the framework's metadata flow [1].

Second, the standard execution path is designed around GPU-resident processing after initial data transfer at the DataPreprocessor stage [5]. Repeated transfers between host and device would introduce avoidable overhead and interfere with throughput-oriented evaluation. For an integrated optical degradation method, this makes preservation of GPU residency a central data-path constraint. [1]

The offline workflow of IDA

In the current IDA workflow, image files are decoded one-by-one on the CPU and transferred to GPU-memory, where they are represented as CUDA-Arrays and accessed through CUDA-Textures during degradation (Fig. 4) [1].

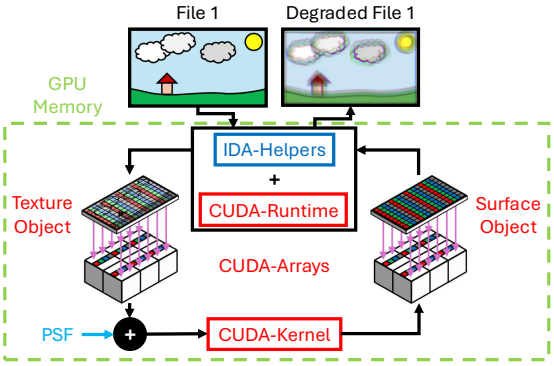


Figure 4: *Offline workflow for applying IDA to an input image [1].*

The result is written to a target CUDA-Array via a CUDA-Surface and then transferred back to host memory for storage as an output file [1]. This offline approach provides a useful reference implementation, but its interface and execution model differ substantially from those of MMDetection.

IDA is focused on explicit per-image file-only processing and external management of degradation variants, whereas MMDetection expects framework-internal propagation of regular, non-pertubated batch data. These differences define the baseline interface and data-flow constraints that must be addressed by an integrated solution.

Design and Implementation

Guided by the integration requirements, CIDPL realizes physically grounded optical degradation as a framework-integrated, GPU-resident extension of the MMDetection data path. Fig. 5 summarizes the resulting end-to-end architecture.

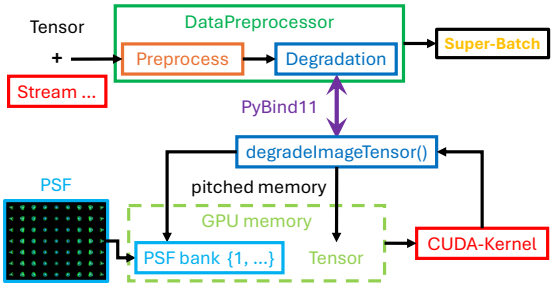


Figure 5: *Complete overview of the CIDPL integration architecture.*

The key idea is to adapt MMDetection GPU-tensors into a CUDA-backend-compatible representation first (*Preprocessing*), then invoke degradation through a functional interface inside the DataPreprocessor without detouring through intermediate image files or CPU memory. The DataPreprocessor is the ideal integration point for this operation, as it is the last step before model forward execution, but the first step to receive batched input samples as GPU-resident tensors allowing parallel processing. Resulting variants are then organized into a traceable, pipeline-compatible Super-Batch that preserves degradation identity. [1]

Preparing GPU-tensors for the CUDA-side

The first integration step is not the optical degradation itself, but the preparation required to make tensor data interpretable by the CUDA backend. MMDetection provides tensor data of shape (N,C,H,W) [10], with each image internally organized in channel-first order. In contrast, the CUDA backend of IDA is designed around image objects with memory organization resembling interleaved RGBA storage, similar to the CUDA-Array. To restore this similarity, each input image is rearranged from (C,H,W) into an interleaved (H,W,C) layout and extended to RGBA format. In addition, the row layout must satisfy pitched-memory alignment constraints. [1]

Pitched memory enables direct CUDA-side access to custom, tensor-backed image data instead of default CUDA-Arrays. Because underlying memory structure of the data is unknown, it must meet alignment requirements for correct pointer access. For this reason, the interleaved tensor representation is padded row-wise where necessary, so that the byte size of each row is a multiple of the required alignment. [9, 32, 1] Fig. 6 illustrates this padding step.

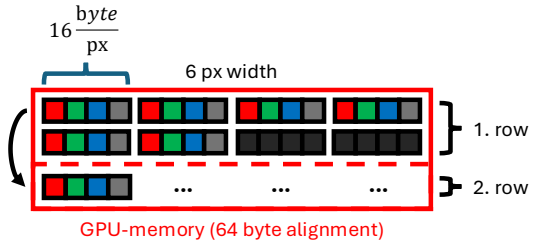


Figure 6: *Example of data padding for pitched memory alignment. For an example image width of 6 px and an alignment of 64 bytes, 2 additional pixels must be padded to each row at float32 precision. [1]*

The padded region is purely technical and is not interpreted as valid image content during degradation. Accordingly, the backend must be able to access custom tensor-backed image data directly. [1] PyBind11 therefore provides the functional and language bridge between Python and C++ [26].

Performing inline degradation on GPU tensors

Once the input tensor has been transformed into a backend-compatible layout, degradation itself is run through the PyBind11-exposed function `degradeImageTensor`. This function is the central integration ingredient of CIDPL, as it exposes the degradation functionality of the modified IDA to Python and makes PSF-based degradation invocable inside MMDetection by a single function call. [1] This design decision is important for two reasons.

First, it turns the formerly file-oriented degradation process into a framework-internal operation. Second, it keeps degradation at the same abstraction level as the rest of the MMDetection preprocessing pipeline. Degraded results can then be converted into a Super-Batch representation and forwarded further. Fig. 7 shows the internal data path for the central function.

The DataPreprocessor passes the prepared tensor through the PyBind11 bridge into the modified C++ backend. There, the tensor is degraded using the existing CUDA-logic and written back into a pre-allocated output tensor that remains resident in GPU memory. [1] Because CUDA-side write access in the established

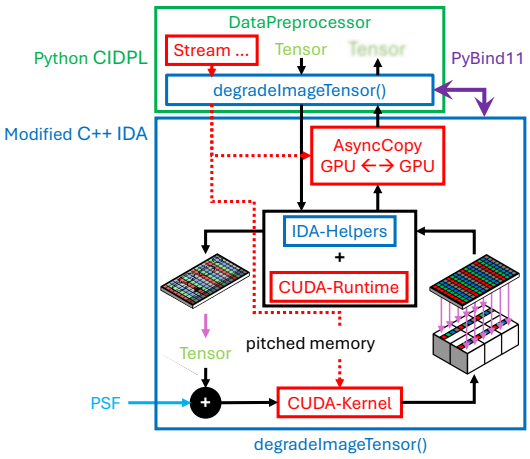


Figure 7: *Internal data path for the degradeImageTensor function [1].*

implementation still relies on the target CUDA-Array and CUDA-Surface path, and pitched memory write access to surfaces is not possible [30], the result cannot be written directly into the final tensor buffer. Instead, the target image is first produced through the established CUDA-side mechanism and then transferred back into the tensor representation. This staged write-back preserves compatibility with the backend logic while still keeping the full processing chain GPU-resident. [1] The same mechanism also supports asynchronous execution.

GPU-to-GPU copies and kernel execution are coordinated through CUDA-Streams, and the stream information originating from the MMDetection-side execution context is propagated into the degradation call, enabling batch parallelism. This allows the integrated pipeline to remain stream-aware, throughput-oriented and real-time instead of degenerating into a sequence of blocking, per-image operations. [29, 1] With all aforementioned optimizations, CIDPL is not merely a wrapper around IDA, but a targeted adaptation of the original backend to the data-flow, interface, and throughput requirements identified in the introduction and background.

The multi-variant Super-Batch structure

Once degradation can be executed, the remaining challenge is to represent and propagate multiple degraded variants per original sample through MMDetection in a structured, traceable and efficient manner. This is required because robustness evaluation against optical degradation typically does not target only one PSF-grid, but a set of grids describing different lens states (e.g. defocus positions) or degradation conditions. A conventional batch representation is not sufficient for this purpose, because it does not explicitly preserve the relationship between one original sample and its multiple degraded counterparts. [1]

To address this, CIDPL introduces the Super-Batch as a structured container for multi-variant sample propagation. Conceptually, the Super-Batch extends the ordinary batch notion by grouping several degraded realizations of the same input batch into one higher-level batch object. [31, 1] Fig. 8 illustrates the resulting data organization.

Let d denote a single degradation variant and D the full set of degradation variants applied to the batch. For each $d \in D$, one ordinary batch of size N is formed. A standard batch of N samples

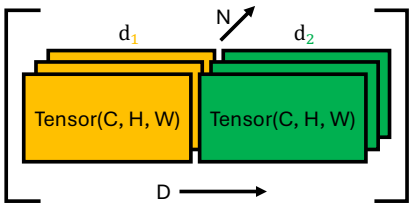


Figure 8: *Conceptual view of the Super-Batch.*

is not flattened across all variants, but replicated at the batch level. The Super-Batch then collects these per-variant batches into one higher-level container. [1] This organization scheme is optimal in the integrated setting for three reasons.

First, it respects the configured batch size N and therefore the corresponding GPU-memory budget. The detector still processes ordinary batches of size N, rather than a flattened tensor with effective size $D \cdot N$, as an alternative representation would require. Second, it allows the model to process each degradation variant batch independently, which is important for both inference-time robustness evaluation and training-time gradient accumulation. [1] Third, it remains conceptually very close to the original MMDetection data format, because the representation is not replaced by a fundamentally new tensor convention, but repacked into one higher-level object whose inner elements are still standard MMDetection-style batch dictionaries. An alternatively considered, yet incompatible, five dimensional representation of shape (D, N, C, H, W) deviates too strongly from both MMDetection and the underlying PyTorch data path. Fig. 9 illustrates the Super-Batch representation in the concrete context of MMDetection.

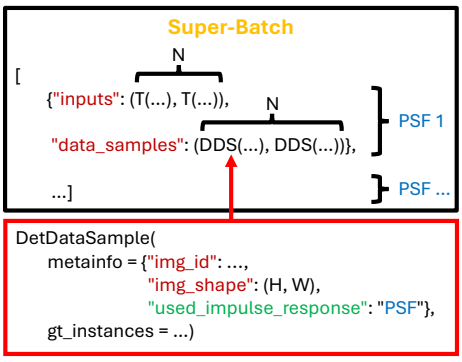


Figure 9: *Super-Batch representation within MMDetection.*

Each element of the Super-Batch contains the usual pair of batched image tensors (T) and corresponding DDS objects. At the tensor level, the inner `inputs` entry remains a standard image batch. At the metadata level, the DDS objects are cloned accordingly and enriched with degradation-specific information. In particular, the identifier of the applied PSF-grid is written into `metainfo`, e.g. through the field `used_impulse_response`. Thus, the degradation identity is not only implied by position inside the Super-Batch, but also stored explicitly in the sample metadata. This makes the format suitable for later back-tracing of predictions to both the original sample and the specific degradation variant that produced them. Based on this representation, following pipeline components can then unpack, forward, evaluate and sort Super-Batch contents in a controlled manner. [1]

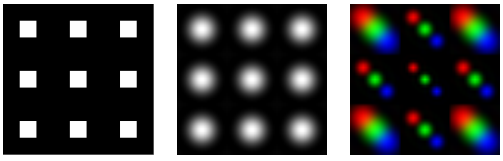
Evaluation procedures and results

This section presents the evaluation procedures along with their results, verifying the initially claimed optimizations, numerical correctness, optical validity, and practical usability of CIDPL.

Numerical validation against the original IDA

A dedicated integration test is required because CIDPL does not simply call the original IDA, but adapts the degradation pipeline for direct use inside MMDetection. Modifications such as the tensor-based GPU-memory access using tensor layout adaptations and pitched memory could introduce unintended numerical deviations at system level. An end-to-end comparison against the original IDA is therefore needed to verify that the integrated CIDPL reproduces the same degradation behavior under identical conditions. [1]

For this purpose, degraded output images of CIDPL are compared directly to original IDA degradations for identical input images and identical PSF-grids. Numerical agreement is quantified by a dataset-level Mean Squared Pixel Error (MSPE) between both outputs, averaged over 100 test images. Two test sets containing synthetically generated RGB noise images with resolutions ranging from 640×480 to 1920×1080 are used to capture variety in image content and resolution. These sets differ in the file format used, namely compressed JPEG and lossless TIFF, to examine potential file-format effects on numerical agreement. [1] Three PSF-grids are used to verify consistency (Fig. 10).



(a) Delta (b) Blurry (c) Variancy

Figure 10: *PSF-grids used for the integration test [1].*

The Delta grid represents a delta function without optical degradation and serves as a control case for isolating non-degradation effects (e.g. library differences). Blurry introduces uniform blur, while Variancy applies spatially varying blur with channel-dependent shifts. The latter is used to verify correct handling of the tensor preprocessing stage with channel interleaving. [1] Resulting MSPE values are summarized in Table 1.

Table 1: MSPEs between CIDPL and IDA for JPEG and TIFF inputs [1].

Input format / PSF	Delta	Blurry	Variancy
JPEG	5.1791	1.4338	4.5271
TIFF	0.0	0.0	0.0

For TIFF inputs, all variants yield a MSPE of 0.0, demonstrating perfect agreement between CIDPL and the original IDA, confirming that the integration does not introduce numerical artifacts. JPEG inputs exhibit small deviations, likely caused by lossy compression and differences in encoding parameterization. The lowest MSPE occurs for Blurry, which suppresses compression noise through low-pass filtering, whereas Delta preserves these differences and therefore yields the highest MSPE. [1]

Optical validation via slanted-edge SFR

Since CIDPL is intended to bring physically realistic degradation into MMDetection, it must be shown that optical behavior is not only represented qualitatively, but also the corresponding transfer characteristics. For this reason, optical validation is performed according to ISO 12233 [16] by qualitatively comparing SFR curves measured from CIDPL-degraded slanted-edge charts against KrakenOS-simulated ground-truth SFR data for the same lens and defocus settings. KrakenOS is chosen as a reference because it is a widely used and validated [17] optical simulation pipeline with tight Python integration. The comparison is carried out exemplarily for a Laikin Wide 100 Degree lens [3] at the center field with defocus positions $\Delta z = 0\,\mu$m, $+50\,\mu$m and $+100\,\mu$m. Spatial frequency is normalized by the Nyquist frequency f_ν. The validation workflow is summarized in Fig. 11.

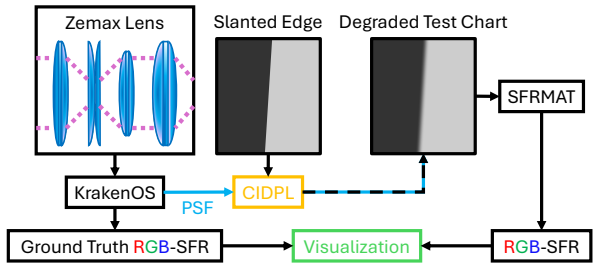


Figure 11: *Workflow for optical validation against KrakenOS ground truth.*

A slanted-edge test chart with an edge angle of 5.71° [16] is degraded in CIDPL using PSF-grids obtained from the KrakenOS simulation of the Zemax lens model. The resulting degraded chart is then evaluated with the SFRMAT library [6] to recover the RGB-SFR curves ($\lambda = \{656\,nm, 588\,nm, 486\,nm\}$), which are compared against the KrakenOS SFRs. Fig. 12 (next page) shows the resulting SFR curves for all three defocus positions.

The SFR curves show close agreement between CIDPL and KrakenOS across all three defocus positions. At $\Delta z = 0\,\mu$m, the curves nearly overlap, while at $+50\,\mu$m and $+100\,\mu$m both methods exhibit the expected monotonic SFR reduction with only larger deviations at higher normalized frequencies, where CIDPL retains slightly more contrast in some channels. Overall, the consistent curve shapes and defocus-dependent trends indicate that CIDPL preserves the relevant optical transfer behavior of the imported PSFs-grids. Thus, CIDPL can be regarded as an optically credible simulator of PSF-incuded effects, suitable for physically grounded robustness studies.

Effectiveness of throughput optimizations

The throughput test quantifies whether the optimizations in CIDPL translate into a practically relevant speedup over IDA and make physics-based optical degradation suitable for real-time use in MMDetection. Throughput is measured in Frames Per Second (FPS) on 1000 random noise images with the same spatial dimensions as in the integration test. Four PSF-grid configurations are evaluated: (1) 5×5 15^2 px, (2) 16×16 20^2 px, (3) 37×37 50^2 px, and (4) 48×27 150^2 px. They were chosen to span different performance regimes, as small PSFs favor stream-level parallelization, whereas large PSFs shift the bottleneck toward raw GPU compute, making multi-GPU execution more relevant. [1]

The experiments are conducted on a RTX A6000 platform. Since IDA only supports single-GPU execution, it is evaluated

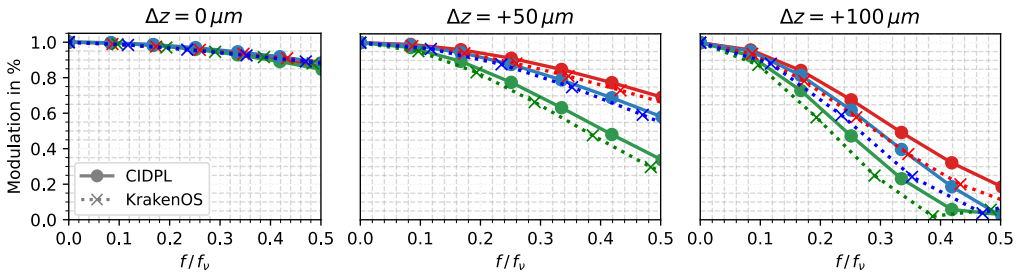


Figure 12: *Comparison of SFR curves for the Laikin Wide 100 Degree lens at defocus positions $\Delta z = 0\,\mu\text{m}$, $+50\,\mu\text{m}$, and $+100\,\mu\text{m}$ for CIDPL and KrakenOS.*

in this mode only, whereas CIDPL is measured in both scenarios with batch size 8 [1]. Table 2 reports the achieved FPS values.

Table 2: Throughput results in FPS between IDA and CIDPL.

Pipeline / PSF	1	2	3	4	Mean
IDA	13.37	14.46	8.88	1.69	9.59
CIDPL (1×)	82.31	74.20	19.77	2.60	44.72
CIDPL (2×)	132.19	116.87	36.39	5.09	72.64

Across all tested grid configurations, CIDPL clearly outperforms IDA. On average, CIDPL achieves a speedup of approximately 4.7x in single-GPU usage over IDA and approximately 7.6x for the dual-GPU configuration. The per-case trend matches the expected behavior with smaller PSFs benefitting strongly from increased parallelism, whereas larger PSFs reducing absolute throughput for all methods, benefitting most from multi-GPU execution.

Overall, results show that the throughput optimizations are practically significant. CIDPL moves optical degradation from an offline preprocessing bottleneck toward direct in-pipeline use. Since IDA requires additional transfer and dataset preparation steps before usage in MMDetection, practical benefits of the integration increase speedup even beyond the shown improvements.

Verifying detection-level effectivity

To assess whether CIDPL is useful beyond system-level integration, we exemplarily conducted fully automated and reproducible detection studies on KITTI [12]. Firstly, training and validating Faster R-CNN [27] across different defocus levels took roughly 10 hours runtime on dual RTX A6000, giving quick and relevant insights for safety-oriented perception research (Tab 3).

Table 3: Test mAP across trained Laikin defocus variants (µm).

Test / Train	Base	-20	-10	0	10	20
Base	0.55	0.59	0.58	0.58	0.55	0.56
-20	0.53	0.60	0.59	0.59	0.57	0.58
-10	0.53	0.60	0.59	0.59	0.57	0.58
0	0.52	0.59	0.59	0.59	0.57	0.58
10	0.52	0.59	0.59	0.59	0.57	0.58
20	0.50	0.59	0.59	0.58	0.56	0.57

A detector trained only on baseline imagery (*Base*) achieves best performance on the baseline domain (0.55 mAP), but drops significantly under defocus mismatch. In contrast, detectors trained on the defocus variants reach consistently higher and more stable performance across all evaluation domains. Importantly, degraded-data training even exceeds the baseline-trained detector on clean baseline data. The second study in Fig. 13 shows the same trend for both Faster R-CNN and Align-DETR [7].

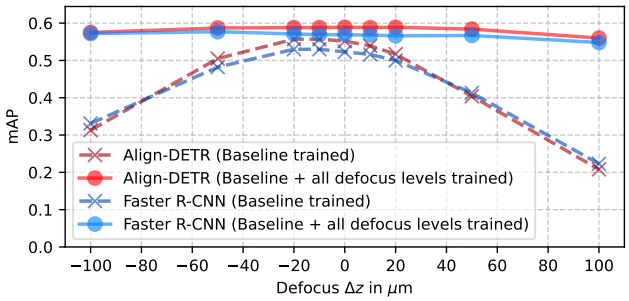


Figure 13: *Laikin defocus sweeps of Faster R-CNN and Align-DETR.*

Joint training with defocus variants strongly lifts mAP across a wide defocus interval instead of improving only a narrow operating point. These results show that CIDPL enables practical robustness studies and helps identify training strategies that reduce optics-induced safety-critical performance loss.

Discussion and conclusion

This paper presented CIDPL, the first practical solution for real-time, PSF-based optical degradation in MMDetection. Combining PyBind11, GPU-resident tensor processing, and the traceable Super-Batch representation, CIDPL turns a previously offline optics workflow into a reusable, high-throughput framework component. Results show that the presented integration strategy preserves numerical correctness and optical credibility while enabling detector-level robustness studies under controlled optical conditions. Importantly, the achieved throughput is above 100 FPS, rendering the degradation process negligible compared to training or inference speeds (on the order of 5 FPS).

Defocus-based training and evaluation reveal robustness trends that are difficult to study in conventional offline workflows, including improved cross-domain stability and, in some cases, gains over baseline-only training. CIDPL therefore establishes a foundation for integrating physically realistic optical effects into routine deep-learning workflows beyond heuristic augmentation or prefabricated datasets.

The study remains limited to one lens family, selected defocus settings, and a restricted set of models and datasets. Future work should extend the analysis to broader lens classes, additional model architectures, and more diverse evaluation scenarios.

References

[1] Maximilian Dornik. *Integration einer CUDA-basierten Bilddegradierungsmethodik in das MMDetection-Framework zur systematischen und effizienten Analyse der Auswirkungen physikalisch realistischer optischer Artefakte auf Objekterkennungssysteme mittels Punktspreizfunktionen*. 2026.

[2] *Abstract Data Element — mmengine 0.11.0rc0 documentation*. URL: https://mmengine.readthedocs.io/en/latest/advanced_tutorials/data_element.html.

[3] *Ansys Zemax OpticStudio | Optical Design and Analysis Software*. URL: https://www.ansys.com/products/optics/ansys-zemax-opticstudio.

[4] Julian Barthel. *Hardwarebeschleunigte Bilddegradation für die echtzeitfähige Simulation von Kameraobjektiven*. 2024.

[5] *BaseDataPreprocessor — mmengine 0.10.7 documentation*. URL: https://mmengine.readthedocs.io/en/v0.10.7/api/generated/mmengine.model.BaseDataPreprocessor.html.

[6] Peter Burns. *SFRMAT 5 - Burns Digital Imaging*. URL: http://burnsdigitalimaging.com/software/sfrmat/.

[7] Zhi Cai et al. *Align-DETR: Enhancing End-to-end Object Detection with Aligned Loss*. 2024. URL: https://arxiv.org/abs/2304.07527.

[8] Kai Chen et al. *MMDetection: Open MMLab Detection Toolbox and Benchmark*. June 17, 2019. DOI: 10.48550/arXiv.1906.07155.

[9] *Custom C++ and CUDA Operators — PyTorch Tutorials 2.10.0+cu130 documentation*. URL: https://docs.pytorch.org/tutorials/advanced/cpp_custom_ops.html.

[10] *DataPreprocessor Source code — mmengine 0.10.7 documentation*. URL: https://mmengine.readthedocs.io/en/stable/_modules/mmengine/model/base_model/data_preprocessor.html.

[11] *Dataset and DataLoader — mmengine 0.11.0rc0 documentation*. URL: https://mmengine.readthedocs.io/en/latest/tutorials/dataset.html.

[12] Andreas Geiger et al. "Vision meets Robotics: The KITTI Dataset". In: *International Journal of Robotics Research (IJRR)* (2013).

[13] Joseph W. Goodman. "Introduction to Fourier optics". In: 1969. URL: https://api.semanticscholar.org/CorpusID:118908270.

[14] Dan Hendrycks and Thomas Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *ICLR* (2019). URL: http://arxiv.org/abs/1903.12261.

[15] Joel Herrera et al. "KrakenOS: Python-based general exact ray tracing library". In: *Optical Engineering* (2022). DOI: 10.1117/1.OE.61.1.015101.

[16] International Organization for Standardization. *ISO 12233:2024 — Digital cameras: Resolution and spatial frequency responses*. https://www.iso.org/standard/88626.html. 2024.

[17] Daniel Jakab et al. "SOLAS 1.1: Automotive Optical Simulation in Computer Vision". In: *IEEE Open Journal of Vehicular Technology* 7 (2026), pp. 179–193. DOI: 10.1109/OJVT.2025.3640419.

[18] Daniel Jakab et al. "SOLAS: Superpositioning an Optical Lens in Automotive Simulation". In: *Electronic Imaging* 37.15 (Feb. 2, 2025), pp. 101–106. DOI: 10.2352/EI.2025.37.15.AVM-101.

[19] Daniel Jakab et al. "Surround-View Fisheye Optics in Computer Vision and Simulation: Survey and Challenges". In: *IEEE Transactions on Intelligent Transportation Systems* (2024).

[20] Masanari Kimura. "Understanding Test-Time Augmentation". In: *Neural Information Processing*. Springer International Publishing, 2021, pp. 558–569. DOI: 10.1007/978-3-030-92185-9_46.

[21] Claudio Michaelis et al. "Benchmarking Robustness in Object Detection: Autonomous Driving when Winter is Coming". In: (2019). URL: http://arxiv.org/abs/1907.07484.

[22] *Model — mmengine 0.11.0rc0 documentation*. URL: https://mmengine.readthedocs.io/en/latest/tutorials/model.html.

[23] Dara Molloy et al. "Analysis of the impact of lens blur on safety-critical automotive object detection". In: *IEEE Access* (2024).

[24] Patrick Müller, Alexander Braun, and Margret Keuper. "Examining the Impact of Optical Aberrations to Image Classification and Object Detection Models". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 48.3 (2026), pp. 2139–2153. DOI: 10.1109/TPAMI.2025.3622234.

[25] Patrick Müller, Matthias Lehmann, and Alexander Braun. "Simulating tests to test simulation". In: *Electronic Imaging* 32 (2020), pp. 1–8.

[26] *pybind11 documentation*. URL: https://pybind11.readthedocs.io/en/stable/index.html.

[27] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: (2015). URL: http://arxiv.org/abs/1506.01497.

[28] *Runner — mmengine 0.11.0rc0 documentation*. URL: https://mmengine.readthedocs.io/en/latest/design/runner.html.

[29] *Stream — PyTorch 2.10 documentation*. URL: https://docs.pytorch.org/docs/stable/generated/torch.cuda.Stream_class.html.

[30] *Surface Object Management — CUDA Runtime API documentation*. URL: https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__SURFACE__OBJECT.html.

[31] *Test time augmentation — mmengine 0.11.0rc0 documentation*. URL: https://mmengine.readthedocs.io/en/latest/advanced_tutorials/test_time_augmentation.html.

[32] *Texture Object Management — CUDA Runtime API documentation*. URL: https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__TEXTURE__OBJECT.html.

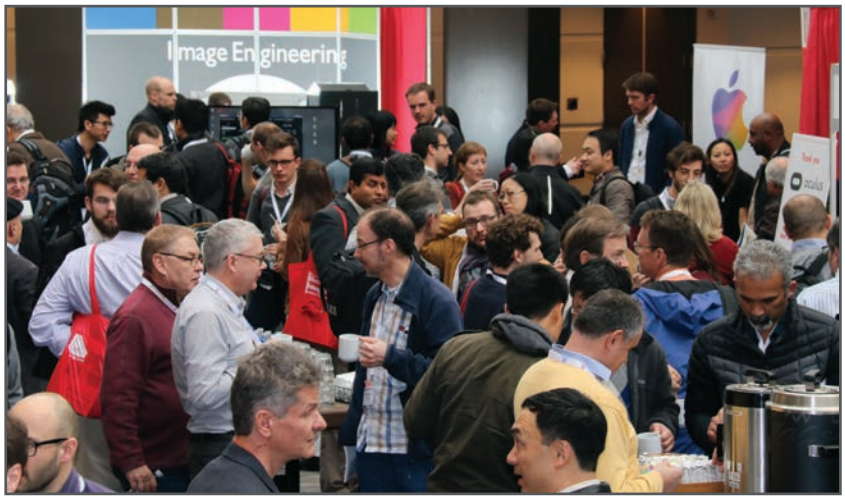
[33] Christian Wittpahl. *Einsatz des mehrdimensionalen Superpositionsintegrals zur physikalisch realistischen Bilddegradation*. 2020.

[34] Christian Wittpahl et al. "Realistic Image Degradation with Measured PSF". In: (2018). URL: http://arxiv.org/abs/1801.02197.

JOIN US AT THE NEXT EI!

electronic IMAGING

Imaging across applications . . . Where industry and academia meet!



• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

ADVANCING IMAGING SCIENCE 75 1947 IS&T 2022 imaging.org