KARAM Christophe

M2 SIGMA

DeepRed by Fondation Grenoble INP
INPG Entreprise S.A. - 3 Parvis Louis Néel - CS 50257 - 38016 Grenoble Cedex 1
Lynred - 364 Avenue de Valence - Actipole CS 10021 - 38113 Veurey-Voroize

Object Detection in Infrared Images

Assessing the viability of the image processing pipeline in a deep learning-based object detection framework

from 01/02/2022 to 31/07/2022

Under the supervision of:

- **Grenoble INP Supervisor: Jocelyn CHANUSSOT, jocelyn.chanussot@grenoble-inp.fr**
- **Lynred Supervisor: Jessy MATIAS, Jessy.Matias@lynred.com**
- **Phelma Tutor: Ronald PHLYPO, Ronald.Phlypo@grenoble-inp.fr**

Confidentiality: No

Ecole nationale supérieure de physique, électronique, matériaux

Phelma
Bât. Grenoble INP - Minatec
3 Parvis Louis Néel - CS 50257
F-38016 Grenoble Cedex 01

Tél +33 (0)4 56 52 91 00
Fax +33 (0)4 56 52 91 03

http://phelma.grenoble-inp.fr

This page is unintentionally left blank.

Acknowledgments

I would like to start out by thanking Jocelyn Chanussot for his support throughout this thesis and beyond. His expertise has helped focus this work and prevent it from degenerating into a giant pool of unguided experiments, putting effort where it really matters. Discussions with his team of students and researchers have helped me stay on top of state-of-the-art object detection, dive into the intricacies of actually training a neural network, and most importantly, embrace common french-isms for technical jargon. Thanks goes out to Paul Vandame, Alexis Noé, and Pierre Falconnier.

I would also like to thank Jessy Matias from Lynred for his untiring efforts in answering my questions about Lynred's datasets and processing pipeline, guiding every step of the experimentation process, offering technical help, replicating experiments to confirm hypotheses and suspicions, and being always available. I am convinced that if all collaborations were of this caliber, the cross-section of industry and research would be much larger.

Finally, I would like to thank the open-source computer vision community for the Herculean task it's undertaking: keeping up with the current state-of-the-art with cutting-edge tools, and always staying ahead of the curve. Needless to say that this work would not have been possible without the community's vast contributions to the field. These contributions have been cited where it seemed appropriate.

Disclaimer

This is a brief disclaimer regarding some deliberate choices that go slightly against the official rules for writing a technical internship report. The guidelines state that the style should be neutral and impersonal. The style of this report is somewhat impersonal, but it does make good use of the "we" pronoun, which clearly goes against the stated guideline. However, this was deemed appropriate by the author (myself?). One of the reasons is that a style can be too impersonal so as to introduce plenty of convoluted sentences in the passive voice, which can sound off. This sentiment appears to be shared by IEEE authors since the first-person is found in most articles [1], and by an article in IEEE's Professional Communication Society that nicely explains the origin of the passive-voice rule in scientific writing: "This "rule" arises from the goal of objectivity and reproducibility in science and engineering, and makes its way particularly into the methods section of technical documents. By eliminating the actor (scientist or engineer) from the text, we can, especially in describing methods, create the (false) sense that these tasks were objectively enacted rather than performed by a subject. Methods written in the passive seem more objective, even as they necessarily imply a subject, because this actor is hidden in the sentence." [2] Besides the use of the plural first-person pronoun, the most used self-reference is "this work". We hope this is not a major inconvenience.

The style of this report is largely neutral except in some cases where it was deemed appropriate to stray away from this neutrality for a more accurate portrayal of certain outcomes or experiments. The report is written in a more casual style than a strict technical report, as it allows us to convey the spirit with which the experimentation was performed, since this is an experiment-heavy project, and this can serve as a reference for any future work that might rely on it. Although statements like "it's hard to setup this experiment" may not belong in a technical report, it sure can help explain the thought process behind this work for anyone truly interested in replicating it and building upon it.

Contents

Glossary

IR: Infrared.
RGB: Red-Green-Blue colorspace, also used to denote images in the visible domain under this colorspace representation.
FP: Floating Point, used in the context of number representation formats such as FP32 or FP16.
DL: Deep Learning.
CV: Computer Vision.
GPU: Graphics Processing Unit, capable of fast parallel computation, speeds up matrix multiplication used in neural networks.
SOTA: State-of-the-art.
CNN or ConvNet: Convolutional neural network, widely used in computer vision tasks.
YOLO: You Only Look Once, state-of-the-art object detection model.
AP: Average Precision, standard object detection metric.
COCO: Common Objects in COntext, widely used dataset for object detection benchmarks.
VRU: Vulnerable Road Users, name of Lynred's dataset, and main dataset used in this work.
Pipe: Short for pipeline, mainly referring to the raw infrared image correction pipeline.
NUC: Non-Uniformity Correction, main Lynred correction algorithm for raw infrared images from microbolometer sensor.
AGC: Automatic Gain Control, Lynred algorithm for optimizing camera parameters online.
BPR: Bad Pixel Replacement, Lynred algorithm to correct bad pixel values in images.
TM: Tone-Mapping, Lynred algorithm for converting 16-bit images to 8-bit images, used in some plots only.

List of Figures

List of Tables

1 Introduction

Computer vision (CV) has been tightly linked to the artificial intelligence (AI) boom since 2012. In fact, the trigger of this AI boom was the ImageNet dataset and classification challenge, and the models that helped solve it in that year. Since then, the field of computer vision using RGB images has greatly improved, always at the forefront of major breakthroughs in deep learning (DL).

More recently, the applications of Deep Learning to Computer Vision have extended beyond RGB images in the visible range to a wide variety of infrared imagery for different applications. One of these applications is the also-booming automotive industry, where infrared imagery can help in low-visibility situations such as fog and low-light scenarios. In these cases, the visible images can be more strongly affected by the environment, while the infrared cameras provide a certain robustness to these external factors.

However, current infrared sensors are highly sensitive to thermal noise, and often require frequent calibration to remain operational in ideal conditions. This is especially the case for microbolometer-type sensors which are passively cooled, and are therefore even more sensitive to thermal noise, which increases as the camera is being used. To counter these thermal noise effects, several post-processing solutions have been developed to denoise the resulting images. These pipelines are put in place by sensor manufacturers like Lynred, and can generally perform a variety of tasks beyond thermal noise correction. Some of these tasks include bad pixel replacement, spatial and temporal denoising, destriping, reducing flare, and finally, tone-mapping, which is the process of converting the 16bit infrared images to 8bit images which can be read by most monitors.

These pipelines are put in place to optimize the visual quality of the resulting output images. However, for certain applications where humans are out of the loop and the images and the information they bring are processed by a machine, there is a little need for visual quality as we understand it. The goal is instead to re-optimize the pipeline to suit the needs of a machine performing a certain task. In our case, this optimization has as its end goal a neural network performing pedestrian detection.

In that regard, the absolute values of the performance metrics are not crucial, and the goal is not to optimize the performance metrics for the object detection task, but rather to study the differences in the resulting performance metrics after application of different processing pipelines to the raw infrared images.

1.1 Thesis Context

This work is inscribed in the context of a Master's degree in Signal and ImaGe processing Methods and Applications (SIGMA) at Grenoble INP, and more specifically, is within the confines of an industrial collaboration between Fondation Grenoble INP and Lynred (chaire industrielle), with Jocelyn Chanussot as its incumbent. J. Chanussot also happens to be the thesis supervisor for this particular work. This collaboration aims to explore applications of infrared imaging, leveraging state-of-the-art methods to extract information from these images, with the help of Artificial Intelligence, which is why the collaboration is aptly named Deep Red.

Organizationally, there were weekly meetings with the two supervisors to discuss previous results and plan for future experiments. In addition, there was a monthly meeting with all members of the collaboration to discuss more important milestones, which was very interesting because it allowed us to see how our work was meaningful on a bigger scale, as well as benefit from new opinions from other members, and gain new insights from other projects in this collaboration.

2 Literature Review

Since the topic pertains to two major ideas, deep-learning-based object detection and infrared imaging, this section will also be split as such. The first part will deal with common architectures used for the detection task, while the second part will examine the applications on infrared (IR) images.

2.1 Object Detection

2.1.1 History

Object detection is one of the three main computer vision tasks, along with classification and segmentation. While there are object detection techniques that are not based on deep neural networks, these have largely become outdated and outperformed by more recent breakthroughs. Some of these techniques rely on scale-invariant [3], Haar [4], or histogram of gradients [5] features. After the neural network boom of 2012, the landscape changed considerably. In 2014, the first region-based object detection model was released, R-CNN [6], followed closely by the first single-shot detector, You Only Look Once (YOLO) [7] in 2015. Since then, the object detection landscape has been largely divided between two-stage detectors like R-CNN, and single-stage detectors like YOLO. Although these models have evolved over time to yield better performances, like the Cascade R-CNN [8], and the fourth-generation YOLO, YOLOv4 [9], these convolutional networks, or ConvNets, have been the main players in object detection tasks. A notable exception is the recent rise of transformer-based models.

Transformer models have been widely used in natural language processing (NLP) tasks, since their inception with the seminal paper "Attention Is All You Need" [10] in 2017, and subsequent breakthroughs with models such as BERT [11]. Their introduction into the computer vision domain has been quite recent, when the first vision transformer (ViT) became the state-of-the-art for image classification in 2020, on ImageNet [12]. Despite its success on image classification tasks, the transformer architecture still lagged behind ConvNets for more difficult downstream tasks such as object detection and semantic segmentation [13]. Hierarchical transformers using shifted windows, or Swin Transformers, reintroduced the sliding window approach into the text-inspired transformers, thereby leveraging one of the inherent strengths of ConvNets [14].

2.1.2 Model Architectures

The previous subsection defined the main actors in the modern object detection landscape: single-stage detectors, two-stage detectors, and Swin Transformers. This may be slightly confusing as these concepts are not necessarily mutually-exclusive and relate to different parts of the general architecture of an object detector. For example, while an object detection model cannot be single-stage and two-stage at once, it can combine Swin Transformers with two-stage detectors. Let's describe the general architecture of a neural network specialized in object detection, which can be described with three main parts.

2.1.2.1 Backbone: Feature Extraction

The backbone is a neural network specialized in extracting features from the input images. This step where a large part of the difference lies in the current state-of-the-art. By taking into account the spatial information encoded in images, convolutional networks have shown to be effective at extracting features from images and have subsequently been used for most computer vision tasks since the dawn of AlexNet in 2012 [13], [15]. As a result, ConvNets have largely been the dominant backbone used in object detection. However, as previously stated, Swin Transformers are now an integral part of the state-of-the-art, and they are used as replacements for the convolutional networks.

2.1.2.2 Neck: Feature Aggregation

The neck is used for feature aggregation, combining in some way information from feature maps at different stages of the backbone. The goal is usually to improve the model's robustness to scale, be that due different object sizes or different image sizes. For example, this should make it easier for a model to learn the representation of a person from examples where people figure in the foreground and background, yielding large and small representations, respectively, for the same object type or class (person). A common neck architecture is the feature pyramid network (FPN), which constructs feature maps at different scales, and most other widely-used architectures make use of the "pyramid" concept in some way.

2.1.2.3 Head: Object Detection

Finally, the head is the part of the architecture specialized in detecting objects, and include the two main families evoked earlier: single-state and two-stage detectors. Two-stage detectors first propose regions of interest - regions that might contain objects - before extracting features from those regions, and completing the detection by the object class, and the refined bounding box [6], [16], [17]. Single-stage detectors omit the first region proposal step and directly predict a class and a bounding box for detected objects [7], [9].

2.1.3 State Of The Art (SOTA)

After this brief discussion of model architectures, we can now gain a better understanding of the current state-of-the-art, and how to best leverage and combine existing practices to train accurate models.

While image classificaion SOTA is done on ImageNet [18], object detection models are mostly benchmarked on the Common Objects in COntext (COCO) dataset [19]. The current version of COCO was released in 2017, with a training and validation split containing 118,000 and 5,000 images, respectively. The test set contains 41,000 images, in addition to an additional non-labelled split of 123,000 images. The object detection task in COCO focuses on 80 object classes. Reading and dissecting the SOTA for re-use in different applications can be challenging, due to many reasons, which we'll briefly outline below:

- As evident by the rapidly moving pace and number of innovations in the field, the SOTA progresses quickly and it can be hard to keep up with the changes. Even from the start of this thesis (February 2022) to the time of writing (June 2022), the leaderboards have changed quite a bit, albeit without any new major breakthroughs. For example, within the sub-field of Vision Transformers, the recent Swin Transformer breakthrough has led to a proliferation of many publications and projects making use of this architecture.
- Anyone familiar with the research world can understand the struggles involved in producing contributions worthy of publication, and those familiar with the world of deep learning research can also attest to the current publication frenzy, in a seemingly "gold-rush" fashion, where researchers try to leverage minor changes in architectures or training strategies to achieve barely-significant performance gains. In that regard, it can be challenging to filter out the useful ideas that can generalize to different tasks, setups, models, and datasets, from those that only lead to marginal gains in specific situations.
- Although replicating research findings has become easier with more open standards for sharing code and datasets, it can still be tough to sift through differing implementations of a same published model, for example. Although the official implementation, if available, may be the most accurate in terms of fidelity to the research article, it may be poorly integratable into existing frameworks and pipelines, which then triggers a trade-off between implementation fidelity (official implementations) and usability (third-party or framework-specific implementations).

A simplified summary of the SOTA is nevertheless rather easy to state, with a few key players and architectures, some of which have already been stated in previous sections. A reliable and very thorough leaderboard is available at PapersWithCode.

- Swin Transformers are the current SOTA, and while the current transformer-oriented explosion keeps producing new models with varying tweaks, the base is still a vision transformer adapted to complex downstream tasks with the use of a window-based approach. Swin Transformers are thus used as a backbone and can be paired with single-stage or two-stage detectors, but are generally paired with high-accuracy two-stage detectors like Cascade R-CNN [8] for optimal performance. If two-stage detectors are inherently slow, then this problem is even further exacerbated by the addition of huge models like Swin Transformers. This makes it that these models, even if they are the SOTA for this task, can not realistically be used in any application where speed is even marginally important.
- YOLOv4 is another family of models that have achieved SOTA performances. Historically, YOLO models have always been in the lead in terms of speed, but at its time of release, YOLOv4 was the best in terms of speed and accuracy. YOLOv4 includes many model configurations for different sizes (in terms of parameters), which leads to a wide array of available speed vs accuracy tradeoffs. YOLO has generated a few off-shoots as well, notably the YOLOv4-scaled version [20], and more notably, the YOLOR model [21], short for You Only Learn One Representation. The paper claims their new YOLOR model is at the top of the benchmarks in terms of both accuracy and speed, but with a few drawbacks: first, the available code implementation has not been updated to keep up with the new SOTA results, and second, the YOLOR model is based on a multi-task approach, which is useful for COCO since the dataset contains annotations for multiple tasks, but can be less applicable for other custom datasets.

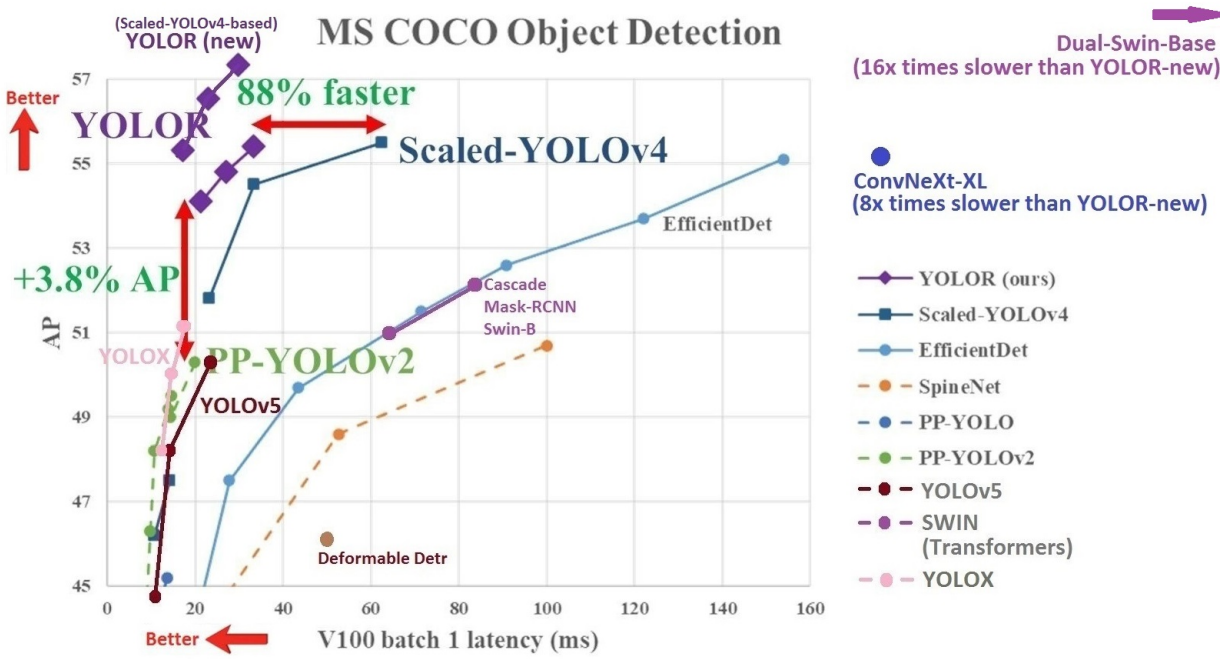

Figure 1: Speed vs Accuracy results for various models on COCO Object Detection, from [22]

2.2 Infrared Imaging Applications

Although developments in computer vision techniques have mainly used images taken by sensors that capture light in the visible range, resulting in grayscale or RGB images, most methods can be transferred to the infrared domain without much loss in efficiency. For example, convolutional object detection models that perform well on RGB images also perform well on infrared images, despite challenges related to low resolution and domain adapation due to the shift in the images' spectral range [23]. Notable applications of object detection to infrared images tackles autonomous driving, where detecting objects in the surrounding can be heavily impacted by environmental conditions. Unlike cameras in the visible domain, infrared sensors are more robust to environmental conditions such as fog, smoke, and darkness [24]. Applications are therefore geared towards improving detection in adverse conditions by combining the information from RGB and IR images. For this purpose, some datasets have been curated and made available to provide a comparable benchmark for developed techniques, most notably, the KAIST Multispectral Pedestrian Dataset [25], offering 95k color-thermal pairs and 103,128 total annotations for the pedestrian category, split into person, people, and cyclist. Another dataset was developed by Teledyne FLIR, a manufacturer of thermal imaging infrared cameras, with 15 classes in 26,442 images [26].

Since the available datasets contain RGB-IR image pairs, most innovative work focuses on methods to fuse information from these two modalities in order to optimize the object detection performance [27], [28]. Hermann, Ruf, and Beyerer have an interesting take on the IR detection problem, whereby they augment the IR images to more closely resemble RGB images. This domain adaptation supposedly bridges the gap between the IR and RGB domains, improving the model's ability to leverage pretraining on RGB images, leading to improvements in pedestrian detection using KAIST.

3 Methods

This section will detail the methods and techniques used in this work. These techniques involve selecting appropriate object detection models from the literature, and such decisions cannot be made without going back to the main objective of this thesis, as defined by Lynred, and refined and focused by the thesis supervisor, in joint coordination.

3.1 Main Objective

The main objective of this work is to optimize the infrared image processing pipeline in order to achieve the best detection accuracy using as little correction algorithms as possible. To put things in perspective, the infrared image processing pipeline contains many algorithms for noise correction and tone-mapping. The current pipeline, as developed by Lynred, is optimized for obtaining the best visual quality from infrared images. In an object detection setting, the images will not be viewed by humans, but rather processed by another algorithm in order to produce the final result: pedestrian detection. As such, there is no need for visual quality as we understand it, and the pipeline should be re-assessed and re-optimized to produce the best possible object detection results. In a way, the pipeline's goal is still to yield the best visual quality, but we are shifting the perspective from humans to object detection algorithms. These detection tasks are meant to be carried out in autonomous driving contexts, and therefore will mainly be executed on smaller embedded devices, where real-time performance is a requirement. For example, an autonomous driving system not operating in real-time is useless for real-world applications (may be useful for research purposes and further improvements, though). From this perspective, we can now better understand the goal of minimizing the image processing pipeline: if certain correction algorithms do not contribute to increasing the performance of a downstream object detection task, then they can be removed from the pipeline in order to reduce computational load and time on embedded chips.

Finally, it is important to stress that under the current objective, the goal is not to optimize the performance of an object detection model, and that the results in absolute values are not important. Rather, we want to examine the relative change in performances with the use of different processing pipelines.

Note: the "processing" pipeline can also be called the correction pipeline, and depending on the context, it can be referred to as a post-processing pipeline (post- image capture), or a pre-processing pipeline (pre- object detection).

3.2 General Pipeline

The workflow adopted in order to achieve this objective contains four main parts, each of which will be detailed in what follows. The input images are raw infrared images encoded with 16 bits. Extracting information from these images is difficult due to the amount of noise present. The first stage of the workflow is the image correction pipeline, whose goal is to reduce the errors from different sources, and obtain a higher-quality image. These corrected images can now serve as an input to train an object detection model, and we get a set of trained weights. These weights are encoded in a 32-bit representation, but can be quantized down to 16-bit, or even 8-bit, representations. This reduces the computational load and memory overhead, thereby speeding up the inference. The final stage of the workflow is to test the different weights on different data sets and evaluate the performance of a given pipeline or setup. These stages are summarized in Figure 2.

3.3 Dataset

The dataset used in this work was provided by Lynred. In some ways, it is similar to publicly available datasets, like the KAIST or FLIR datasets (see Section 2.2), in that it contains RGB-infrared image pairs. The provided dataset contains 6339 pairs of RGB-infrared images with 24715 annotated bounding boxes for the class "pedestrian" or "person", which includes cyclists and bikers. These infrared images are corrected by Lynred using well-calibrated images, and are 8-bit encoded, like regular images. A subset (5944) of these images is provided in raw 16-bit noisy format. This raw subset will be our main starting point, since we care about applying different correction pipes to these raw images. In addition, there are 5720 multispectral images with 23628 annotations, which are false-color composites of fused visible and infrared images. These multispectral images will not be used in this work. An example of annotated images is shown in Figure 3. The dataset is called the VRU dataset, for Vulnerable Road Users, and will be referred to as such in what follows.

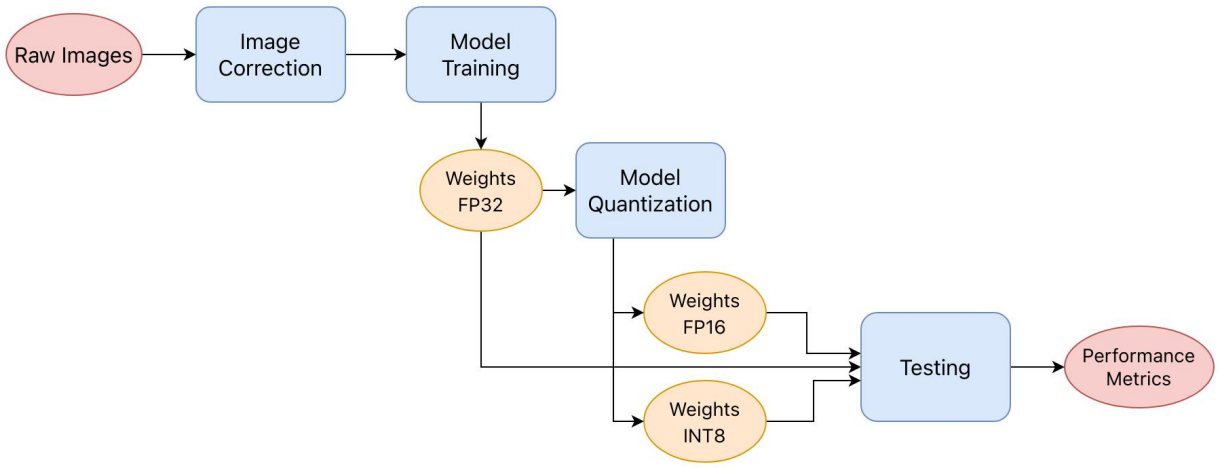

Figure 2: General workflow for optimizing the infrared processing pipeline based on pedestrian detection performance.

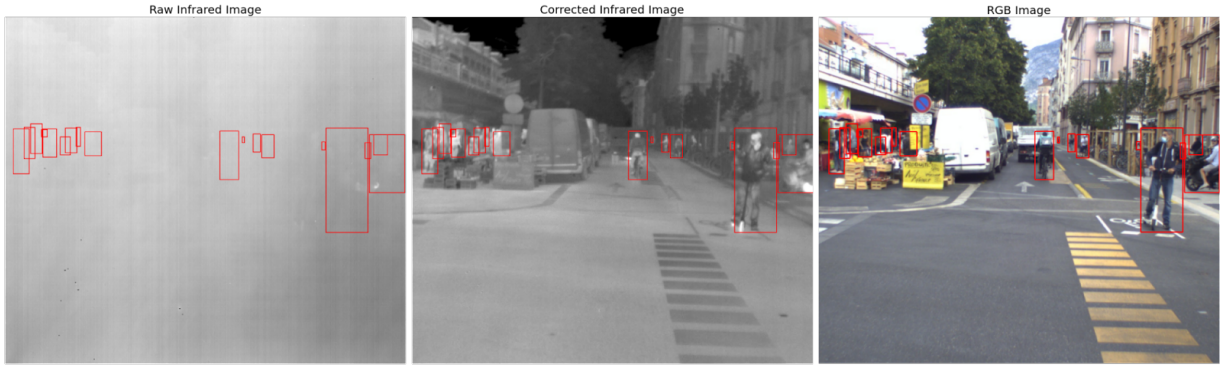

Figure 3: Image with annotated bounding boxes from Lynred VRU dataset across different domains: raw and corrected infrared, and visible RGB.

3.4 Infrared Image Correction

The entire workflow's goal is to optimize the infrared image correction pipeline, which was developed by Lynred. Lynred provides access to these proprietary correction algorithms in the form of a Software Development ToolKit (SDK), which has interfaces in Python, C++, as well as a full-featured graphical user interface (GUI). An overview of the correction pipeline is shown in Figure 4.

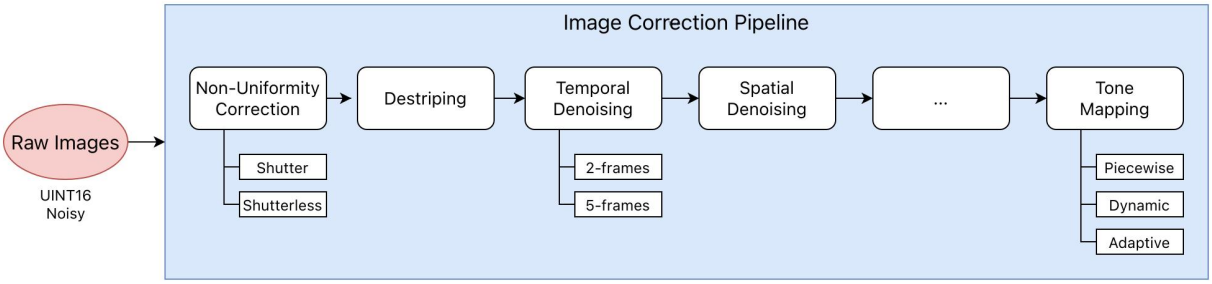

Figure 4: Lynred Raw Infrared Image Correction Pipeline

3.4.1 Non-Uniformity Correction (NUC)

The non-uniformity correction is a correction of the thermal noise in the image, present due to the heating of the electronic elements in the sensor. Although some infrared sensors are actively cooled, these are more expensive and geared toward higher-stakes applications (like military ones). For autonomous driving and consumer-level applications, Lynred uses microbolometer type sensors which are passively cooled. As a result, as the camera is

used, it heats up, and leads to the build-up of thermal noise. For a constant ambient temperature, two pixels in an image do not have the same response, which is an offset error. On top of that, when the ambient temperature varies, the pixel responses do not vary uniformly, which is a gain error. So, if we plot pixel values as a function of ambient temperature for a fixed scene temperature, we have to correct the y-intercept (offset) and the slope (gain) so that all pixels behave the same way, or uniformly, whence the name, Non-Uniformity Correction.

The estimation of the offset and gain correction is done through a calibration step, and the calibration pipes are split into two categories: shutter pipes and shutterless pipes. Shutter pipes require two sets of input images: hot and cold. The hot and cold images are taken at the same ambient temperature with the camera looking at a uniform black body. The shutterless pipe requires three sets of input images: sets of hot and cold images at a given ambient temperature, and another set of either hot or cold images at another ambient temperature. Once the pipe is calibrated, it can then be used for the correction step, or the "execution" step.

3.4.2 Other Denoising Algorithms

Even though the NUC is the largest contributor to the corrected image's visual quality, other algorithms are also involved to correct for different types of noise. For instance, infrared images can have stripes, bad pixels (with unexpected values), flare noise (bright spots), and as such, there are algorithms to correct for each of those. The full list of algorithms includes:

- Bad Pixel Replacement (BPR): replace non-operational pixels that are detected as a non-uniformity in a given neighborhood
- Destriping: removes vertical stripes generated by column noise
- Spatial Denoising: removes high-frequency classical spatial noise
- Temporal Denoising: denoising using the temporal coherence between successive frames
- Automatic Gain Control (AGC): optimizes the camera parameters for a better dynamic range
- Evolpel: removes bad pixels that appear over the lifetime of the sensor

Even though the key player is the NUC, it's these algorithms that are going to play a major role in optimizing the correction pipeline. Because the NUC is absolutely necessary, there won't be much to in terms of alleviating its computational load. However, the big question lies in whether these accessory algorithms, which are important to achieving good visual quality for human perception, are equally important for an object detection task. Each of these algorithms may have one or multiple methods to achieve their target goal, and each of these methods can also have its own set of parameters.

3.4.3 Tone-Mapping

The last piece of this pipeline is the tone-mapping. The raw infrared images are encoded in 16-bit unsigned integer representation (uint16), and the entire correction pipeline operates within this representation. So, at the end of the pipeline, we still have uint16 images. Monitors are not capable of displaying 16-bit images, and so the images we are familiar with are in 8-bit representation. The tone-mapping algorithm takes care of converting the images from 16 to 8 bits. However, as a result of this conversion, there is an irrecoverable loss of information since we have reduced the space available to encode information.

3.5 Object Detection Model Selection

We have previously went over the state-of-the-art for object detection in Section 2.1.3, and we have just reviewed this work's main objective, and only now can we proceed to a reasonable and justified selection of candidate object detection algorithms. Simply put, the main candidates are the Swin Transformer family of models, and the YOLOv4 family of models, as per the SOTA. The YOLOv4 family of models include the sub-families of YOLOv4 [9], YOLOv4-Scaled [20], and YOLOR [21]. Each sub-family contains a number of model configurations for different sizes, but this is less important for now. Due to the drawbacks evoked in Section 2.1.3, the YOLOR family of models will be discarded. The reason are its reliance on dataset with multiple tasks, which cannot be properly leveraged in our case, and the lack of a reliable implementation with the stated results as of yet.

The YOLOv4 official implementation is in its own open-source framework written in C, Darknet [29], same as the previous YOLO versions. The YOLOv4-Scaled official implementations are either in Darknet as well, or in

PyTorch. By using the Darknet framework, we can then have access to official implementations of YOLOv4, and all previous YOLO models. From experience, the Darknet implementation of YOLOv4 is the best one in terms of fidelity to the model (since it's the official one), and speed, since it is written entirely in C, with a command-line interface. While the YOLO architecture has been ported to various frameworks and sub-frameworks, these implementations suffer from the speed limitations imposed by their respective frameworks. To put things in perspective, the Darknet framework was released in 2013, very early relative to current innovations in DL, and predates the official releases of the two most popular frameworks, Tensorflow (2015) and PyTorch (2016). Darknet has thus had the time to mature over the years, always being at the forefront of object detection with 4 generations of YOLO models, which might help explain its reliability, performance, and general appeal.

Swin Transformers are for the most part implemented in PyTorch, especially when it comes to the official versions. Furthermore, MMDetection [30] is an object detection framework that uses PyTorch, and has become somewhat of a standard for training and creating new Swin Transformer models. However, the toolbox contains many more models, which allows for quick benchmarking and testing to find an optimal solution. In this framework, Swin Transformer backbones can be paired with detectors such as Cascade R-CNN or RetinaNet. We can also swap out the transformer backbone for more classical ConvNets, such as ResNets.

Possible simple backbone-head combinations are shown in Figure 5.

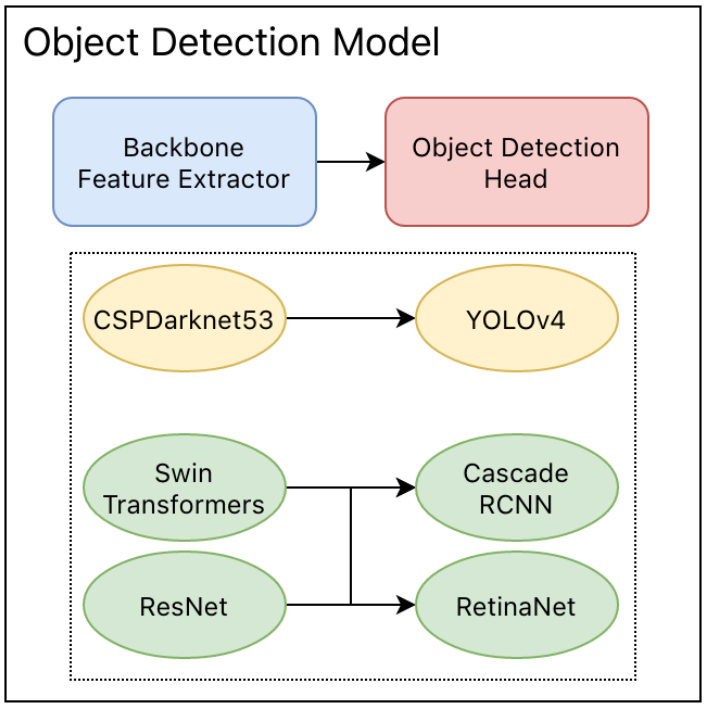

Figure 5: Object Detection Model Architecture

3.6 Weight Quantization

Typically, models are trained using the single-precision float-point format, called float32 or FP32. This means that at the end of training, the produced model weights are in FP32 representation. For inference on embedded devices with more limited resources, it can be beneficial to reduce the precision the model, and deal with half-precision floating points, float16 (or FP16), and even 8-bit integers (INT8). In this work, the weights and model are converted after training, for use in inference only. There are techniques available to perform Quantization-Aware Training (QAT), but this greatly increases the complexity of the experimentation pipeline and is not yet available in all frameworks, so it is not used here. Quantizing the model can help with inference speed due to the reduced computational requirements, and the reduced data bandwidth, meaning that the device spends less time transferring data from memory to computing elements and less time reading and writing data, resulting overall in less power consumption [31], [32].

4 Results and Discussion

4.1 Experimental Setup

4.1.1 Hardware Setup

Any experimentation in deep learning, especially one dealing with high-resolution images, has heavy computational requirements. For this work, all experiments were trained on a machine with an AMD Ryzen 7 5800X 8-Core CPU, 64GB of RAM, and an Nvidia RTX 3060 GPU with 12GB of VRAM. For the embedded experimentation, Lynred provided an Nvidia Jetson Nano for processing, along with one of their microbolometer-type infrared cameras. All results, especially ones related to inference times, are reported for the RTX 3060 GPU, unless otherwise stated.

4.1.2 Software Setup

The software frameworks were chosen based on the model selection discussed previously. As such, three frameworks are selected:

- MMDetection [30], which is an object detection framework built on top of PyTorch. The main reason for using MMDetection is their implementation of the Swin Transformer models, which is used by many recent publications as well as the official Swin Transformer for Object Detection implementation. An added benefit of MMDetection is the availability of more than 350 model combinations and the pretrained weights for the respective backbones, making the task of testing and benhcmarking different models a rather easy one.
- Darknet [29], which is an object detection framework built from-scratch for the YOLO models. It remains the framework with the official implementation of all YOLO models, and one of the fastest ones for model inference.
- TensorRT is a C++ library developed by Nvidia for use in inference on Nvidia GPUs and deep learning accelerators, and this allows us to perform the model quantization and accelerate inference on embedded devices like the Jetson. While the previous two frameworks were deliberate choices, this one was more of a matter-of-fact since we are using an Nvidia Jetson Nano.

On top of these frameworks, this work does a good job of writing a framework-agnostic set of tools written in Python for dataset creation, model training, weight quantization, and inference and testing. These tools can be manipulated through a single configuration file, which makes it easier to run experiments on a massive scale, which is a major part of this work. This overarching custom framework would not have been possible without Docker, a containerization technology that allows, in our case, for easily packaging dependencies at the system-level, making it feasible to manage such differing frameworks.

4.2 Experiment Design

The experimental design for this work regroups many elements, and having any variables to test for it can make the experimental design a rather long one. Generally, these elements are as follows:

- **Object Detection Model:** We have to test for the selected detection model. Choosing between the two families of Swin Transformer or YOLO does not bring us much closer to a specific model. Within each family, there are many options to look out for. For Swin Transformers, this includes the size of the model (tiny Swin-T, small Swin-S, base Swin-B), and the detection head it is paired with. For the YOLO models, this includes many of the default configurations provided with YOLO for adding scaling, changing model size, and other details like activation functions as well.
- **Quantization:** We have to test the performances of each model with the default FP32 weights, then with reduced precision for FP16 and INT8.
- **Testing Sets:** We can test these results on different sets from our main dataset, but we can also test on different datasets from other providers in case this provides additional insights.
- **Hyperparameters:** Even if our main goal is not to optimize performances by fine-tuning hyperparameters, we still might want to try out different parameters that pertain to training, such as the number of epochs, and more importantly, the input dimensions (spatial image size and number of channels).

The items listed above are all mostly secondary parts of the experimentation pipeline. The main part of this process is benchmarking the different processing pipelines to observe their effects on object detection performance. The processing pipelines are split into two families: shutter and shutterless. Each of those families includes multiple algorithms, each with its own methods, and each with their own parameters. Furthermore, the choice of calibration temperatures for each of the pipes is also to be considered. For the shutter pipe alone, testing all possible calibration temperatures, with all other things being equal, can easily give us 40 combinations (for temperatures ranging from 10 to 50C).

Due to the sheer number of variables present in this experiment, we cannot simply brute force our way to an answer, and must consider a careful experimentation based on initial prior knowledge (from Lynred's expertise and best interests), and new information that comes to light throughout the experimentation process. For example, during experimentation, if we realize that a simple benchmark of a processing algorithm seems to conclude that the algorithm brings no substantial effects to the object detection, there is little use in diving deeper and experimenting with varying all possible methods and parameters of that algorithm. Generally, if it doesn't seem to provide benefits, or if it even degrades performances, then it should be dropped. If it provides benefits, then what parameters can be tuned to further optimize these gains?

Throughout the experimentation, it is important to remember that we are optimizing for speed and object detection accuracy, and carefully examining this trade-off for each part of the pipeline.

4.3 Experimental Results

4.3.1 Datasets and Terminology

This section contains results for many experiments, so it's important to keep track of the datasets used and the terminology that goes along with it. The Lynred-provided dataset will be referred to as VRU. The provided dataset contains images in the visible domain (3-channels), referred to as RGB-Base, and in the infrared domain. The images in the infrared domain are either the uncorrected, raw 16-bit single-channel images (IR-Raw), or ones pre-corrected by Lynred (IR-Base). The corrected images are processed through a pipe whose calibration conditions are almost identical to the conditions during which the images were captured, so it is sort of a "gold standard" of correction in terms of visual quality, that we cannot reasonably replicate in our experiments. So, in essence, the RGB-Base and IR-Base datasets will be used to set baseline performances and evaluate parameters independent of the correction pipeline. The IR-Raw images will be fed through various pipes to obtain corrected IR images, which can then be used to evaluate the pipeline.

Most object detection models use image sizes that are multiples of 32. In our case, most IR images are 555×479 ($width \times height$) in size, and most RGB images are 1117×962, natively. This means that they will be used in the network at resolutions of 544×448 for IR and 1088×960 with resizing that preserve aspect ratio.

Early on, we realize that training at higher resolutions takes more time, but usually results in better detections, which isn't very suprising. However, we find that for IR images, training using single-channel inputs leads to poor results compared to training in three-channel. The infrared images are only single-channel images, but can be read as "RGB" by simply duplicating the channel 3 times. This results in slower training and inference and a heavier computational load due to the tripling of the input size, but yields to better results as the model is able to better leverage the pre-training which was done on RGB-images. Therefore, the bulk of the experimentation is done using three-channels, but this fact is kept in mind for further optimization down the line.

4.3.2 Performance Metrics

Since this work is heavily focused on benchmarking and experimentation, it is important to understand how to evaluate the different trials with adequate metrics. In object detection, the most widely metric is the Average Precision, or AP. When it is computed over several classes, it is referred to as a mean Average Precision, or mAP. Generally, the terms are somewhat interchangeable. This average precision metric combines precision and recall: how accurate the bounding boxes are, and how many true bounding boxes are detected. This metric is generally evaluated at different Intersection over Union (IoU) thresholds. In object detection, the IoU threshold decides what counts as a positive detection, and so higher IoU thresholds penalize the model more heavily. A common threshold is 0.5, and the metric is then called AP@.50, and this is the one we'll mainly be using across these results. It is important to note, once again, that we are mostly interested in the relative differences between trial metrics, and not their absolute values. Since we are dealing with different frameworks, the ways of computing the AP@.50 can differ, and we end up with different results. This is why it is important to note than when we mention Training or Validation AP, it is the metric as computed by the training framework (MMDetection or Darknet). When we mention the Test AP, it is the one computed by our framework on the test set, largely based

off of the official COCO implementation for this metric. It is also common practice to separate the performance metrics based on object sizes, which we also show for some trials. These size thresholds are set based on the dataset content and agreement with Lynred.

4.3.3 General Benchmark Overview

To get a general idea of the datasets we're dealing with and the possible processing pipes, we start off with a general benchmark that should provide a good overview and give us some foresight as to how to plan the following experiments. This general benchmark is shown in Figure 6. The chart shows the testing AP@.50, the network inference time, and the pipe correction time, which groups together the three main metrics we care about. We want to optimize the correction pipeline in order to decrease the correction time while increasing the detection accuracy, and we need to choose a model that performs well on baseline sets while still providing adequate portability to embedded devices. We can extract many conclusions from these results.

The first important point to underline is the difference between the three baseline datasets: the RGB images (RGB-Base), the pre-corrected IR images (IR-Base), and the raw images. The use of infrared images for pedestrian detection is justified by the 12 point increase in AP when using IR images instead of RGB images. The use of a correction pipe is also warranted: detection on the raw infrared images produces extremely poor results. Even if the same information is present in the raw and corrected images, the raw image drowns the information in noise and the network ends up not learning much.

Another important point to notice is that the dataset contains very small objects that can often not even be recognized as people by the humans in these infrared images, and they are labeled as such due to having more information in the labeling stage coming from temporal continuity between successive frames and from the visible spectrum in RGB images. Therefore, in most experiments, the datasets have been filtered to no longer include labels for objects under 20px in height, which can be seen by the use of the NS suffix (NoSmall). For a YOLOv4-Tiny-3L model, the main metric improves by 20 percentage points on the test set, from 30% to 50% after this filtering process. While this may filter out a portion of potentially recognizable objects, the limit is set fairly low at 20px, and this allows the performance metrics not be heavily affected by such noise, allowing us to gain a better understanding of the differences between multiple experiments.

After that, we move on to trials using correction pipes on the raw images. The correction pipes are either shutter or shutterless, and they either use 1, 2, or 3 references in the calibration process. The shutter pipe can use 1 or 2 references, while the shutterless pipe can use 2 or 3. Also, the pipes are configured with either all algorithms activated with their default parameters, or all algorithms disabled. We have some interesting results. The shutter pipe performs best when it uses 2 references and no other correction algorithms, while the shutterless pipe performs best when it uses 3 references and all correction algorithms. One conclusion is easy to make, we need as many references as we can get, as this will always improve the quality of the correction for more varied environmental conditions. The second conclusion is a bit tougher, but is guided by Lynred's expertise. The shutter pipe in our case performs decently well to begin with because the calibration conditions are not too far off from the conditions in which the images were taken. In fact, the shutter pipe offers the best possible correction when the calibration step offers the same exact conditions as the capturing environment. However, since this is difficult to obtain in practice, especially for autonomous vehicle applications, where we have to cover a wide range of temperatures from-20°C to 50°C, and maybe even more extreme temperatures. In these applications, the shutterless pipe performs much better, which is why it is more interesting to dive into a deeper exploration of the shutterless pipe rather than the shutter pipe.

Finally, we move to examining the quantization step, which generates satisfactory results. The FP16 quantization results in a drop of 3 points from its FP32 counter-part, but leads to a staggering drop in inference time from 6.71ms to 0.96ms. Quantizing even further to INT8, we lose almost 9% from the non-quantized performances, and we divide the inference time by 10. The FP16 quantization seems like the best trade-off between speed gain and accuracy loss, and since the images and the processing pipe are natively in 16-bits, it is probably more interesting to adopt the FP16 quantization.

4.3.4 Benchmarking YOLO Models

The official YOLO repository makes many model configurations available to the public. YOLOv4 and YOLOv4-Tiny-3L are the "vanilla" YOLOv4 models [9], the former is the standard configuration, and the other is a smaller version of it. The "3L" suffix denotes the three YOLO layers present in the model head. Other configurations such as YOLOv4-CSP and YOLO-P5 are implementations of the YOLOv4-Scaled models from [20]. We train some of these models on the IR-Base dataset, filtered to remove objects whose heights are smaller than 20px. The reasoning for this will be discussed shortly after. Figure 7 shows the mean average precision of various YOLO models on the validation set, as well as the inference times. Inference time results are not surprising.

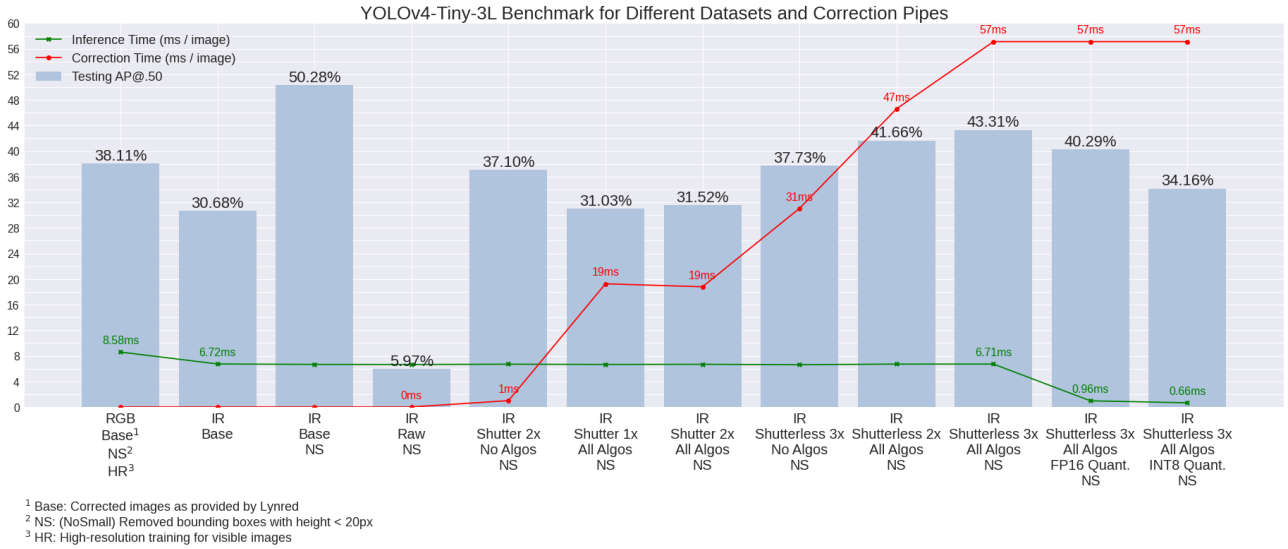

Figure 6: Benchmarking general variations of the processing pipeline and datasets for YOLOv4-Tiny-3L on VRU.

The "tiny" model performs best, beating other models by more than 10ms per image. For the other models, the CSP architecture adds around 1ms of inference time to the vanilla YOLOv4, and the biggest of all the tested models has the largest inference time, at around 24ms. In terms of accuracy, the "tiny" model performs the worst, which is expected, but the other models seem to be almost equal in performance, with the largest model coming in slightly behind. These experiments do not offer much in terms of statistical significance, but allow us to get a general idea of the performances that can be expected. In conclusion, to optimize for accuracy, the choice of YOLO models is not very crucial. To optimize for speed, the choice is obvious: go for the tiny models instead of the larger ones. Another important factor that is missing from the chart is the training time. Training time dictates the number of experiments that can be performed over the course of a given timeframe. Since we have a lot of experiments to run, it is preferable to go for a lighter model so that more experiments can be performed, and so that experiments can be repeated for more statistical significance. Bottomline is that the YOLOv4-Tiny-3L model offers us the greatest flexibility in terms of embeddability with fast inference, experimentation with short training time, all that without sacrificing too much accuracy. This model with then be the one used for multiple experiments thorough experimentation with pipeline configurations.

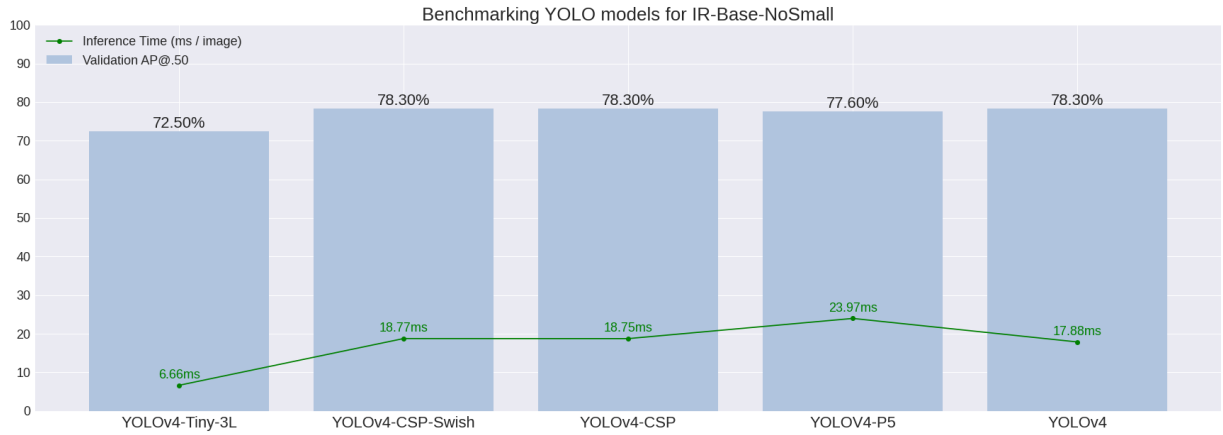

Figure 7: Benchmarking YOLO models

4.3.5 Benchmarking Swin Transformers

Swin Transformers are the new state-of-the-art in computer vision tasks, notably in object detection. The leaderboards move frequently due to the current craze of inserting a Swin Transformer backbone into various architectures and training methods. Within the realm of usable Swin models, the best performing ones in object detection use Cascade R-CNN as the detection head, and this is briefly summarized in Table 1. All models are pre-trained on ImageNet-1K. Most top leaderboard scores use Swin-L as the backbone, an even large model

Backbone	Head	Box AP	# Params
Swin-T	Cascade R-CNN	50.4	86M
Swin-S	Cascade R-CNN	51.9	107M
Swin-B	Cascade R-CNN	51.9	145M

Table 1: Comparison of Swin-Based Cascade-RCNN Models on COCO dataset from [14]

Backbone	Head	Test AP@.50 (%)	Inference (ms)
ResNeXt-50	Faster R-CNN	13.93	**29.6**
ResNeXt-101	Cascade R-CNN	18.91	57.6
ResNeXt-101	VF-Net	18.00	45.5
Swin-B	Cascade R-CNN	**47.62**	56.1
Swin-T	Cascade R-CNN	36.73	34.8

Table 2: Comparison of MMDetection Toolbox Models on VRU-IR-Base-NS

than Swin-B, but that is not available for training on common repositories. The goal of testing these models is because they are the SOTA. This can allow us to evaluate the applicability of SOTA models to real-world scenarios, and if we get better results, we can still use these models to evaluate the validity of different pipelines. However, we do not expect these models to be used for live inference as they are extremely slow and definitely not suitable for production environments where speed is required.

Whereas the testing AP@.50 for YOLO models hovered well above the 60% mark, the best Swin models struggled to break the 50% mark. The best iterations using Swin-T resulted in models with AP@.50 around 37%, which is much less than expected from these SOTA models. At first, it was thought that the implementation was faulty or that our application handled it badly. However, as more experiments were performed with different model combinations, the reasons for the failure of the Swin models became clearer. A Swin-B backbone was much more performant with an AP at 47%, but it was still no where near the small YOLO model. For the full story, we can take a look at the inference times as well. With Swin-T, we have 35ms per image, which is x2 that of YOLOv4, and x6 that of YOLOv4-Tiny-3L. For Swin-B, that goes up to 56ms per image. Replacing the Cascade R-CNN head with RetinaNet results in lower accuracies. To gain a better understanding of the relative shortcomings of these SOTA models, we try them out on a very small dataset, the Balloon dataset, originally used to test Mask R-CNN models. The balloon dataset has a total of 62 training images and 255 fairly large training bounding boxes for one single class. The Swin-B + Cascade R-CNN model achieves 9% after 30 epochs, and 24.1% after 200 epochs. The dataset is small and very easy and yet, our Swin-S + RetinaNet model cannot even get a 1% accuracy after 30 epochs. That same model, when trained for 2 epochs on the COCO dataset with 80 classes and 110K training images, manages a 11.9% mAP. If we filter the COCO dataset and only use the "person" class to match our dataset, we manage to get a 66.2% mAP using Swin-T + Cascade R-CNN after only 10 epochs, as a full training would require more than 3 days. Using the same model and training it on VRU-RGB-Base, we can only get 38% mAP after 36 epochs. Note that 36 epochs is the "full", "3x" training schedule as per the default configuration. In light of these results, the conclusion becomes much clearer: the anecdotal statements about transformers needing a lot of data may be true after all.

Finally, if these Swin models under-perform compared to YOLO models, how do they fare against other candidates in the MMDetection toolbox? The results are summarized in Table 2. Swin models fare very well against the best performers in MMDetection, thereby cementing their SOTA status, but they still cannot compete with the YOLO models in terms of accuracy and speed.

4.3.6 The Role of Tone-Mapping

The raw infrared images are in uint16 representation, and the entirety of the Lynred processing chain, except for the tone-mapping algorithm, preserves this representation. Down-sampling to 8-bits leads to an important loss of information simply for the purpose of displaying images on monitors. From this perspective, Lynred suggested removing the tone-mapping algorithm so that the neural networks can make use of the full dynamic range of the image and receive inputs in 16-bits directly. However, every attempt at getting rid of this tone-mapping has failed.

First, it is important to note that virtually all frameworks for training neural networks operate in float32. It is possible to train in float64 in PyTorch, but that is not interesting to our use-case, and neither is it interesting for most use-cases. Even with the introduction of mixed-precision training (mixing single and half-precision, or fp32 and fp16), it is still not possible to train an entire network in float16, due to some limitations in certain operations, like Batch Normalization. Mixed-precision training is faster because of the lower memory

Method	Average Precision	Standard Deviation
Clahe	68.87	0.82
Dynamic	68.43	0.34
Std3	67.80	0.78
Adaptive	68.37	0.76
Piecewise	**69.20**	0.00

Table 3: Comparison of tone-mapping methods for a shutterless pipe (3 runs)

requirements, and because it can make use of new tensor cores Volta and Turing graphics cards architectures. However, it is not possible, nor desirable, to train a network in int16, as the network will only take float inputs. In most applications, the images are already encoded in 8-bits, and the network pre-processing normalizes the images by dividing by 255, essential a min-max normalization, resulting in images represented as a float between 0 and 1.

Having described how floating point precision affects neural networks, we can then only formulate one possible solution. Instead of converting our infrared images from uint16 to uint8 with tone-mapping, then converting uint8 to a float representation, we can directly feed the networks images in float32, bypassing the tone-mapping step. This raises two challenges: - Since images cannot be saved on disk as a float, in common image formats, the conversion will have to be done on the fly. This poses a light software challenge as we're dealing with larger frameworks like MMDetection and Darknet, but is ultimately easily doable. - The second, and more important challenge, is how do we then convert the images from uint16 to float? The naive way is to simply perform a min-max normalization like the one that is usually done, but instead of dividing by $2^8 - 1 = 255$, we divide by $2^{16} - 1 = 65535$. This is where we quickly run into problems, as the histogram of our infrared images is not straightforward, and is heavily affected by the processing steps in each pipeline. For example, the histogram before bad-pixel replacement is very different from the one after bad-pixel replacement. Another key point is that although the representation covers 2^{16} bits, the actual dynamic range only covers 2^{14}, and even worst, the actual useful dynamic range without bad pixels covers about 2^{10}. This makes it so that any simple processing such as a naive min-max normalization cannot succeed consistently on our set of images. A more complex processing would simply end up being a rewrite of Lynred's functionality, which is not the point. In addition to min-max normalization, z-scoring is also tested for a variety of different setups, aiming to get rid of the tone-mapping by eliminating the need to quantize to 8-bits before getting to floats. The results for all variations were very empirically conclusive: the tested methods do not allow for discarding the tone-mapping algorithm. After min-max normalization, the histogram ends up being very squashed and spread along a vast range, which seems to result in a dilution of the information, with the YOLO models consistently achieving a null accuracy.

We have then concluded that the tone-mapping plays a crucial role in an object detection pipeline and cannot be sacrificed. Lynred's experience shows that the piecewise method of tone-mapping performs the best, and we can verify this experimentally. As seen in Table 3, the piecewise method does in fact perform the best, slightly ahead of other methods which are close to each other, albeit with relatively high standard deviations over 3 runs. The adaptive method was interesting for Lynred since it was still in development, and it appears to be not far off from other methods. The piecewise tone-mapping will then be used for all experiments down the line.

On a final note, in experiments where tone-mapping is not used, the images end up being saved in a uint16 format on disk at the end of the pipeline. By default, the network's pre-processing steps read the images as 8-bit before normalizing it, effectively performing their own version of tone-mapping.

4.3.7 Benchmarking the Shutter Pipe

The shutter pipe is not a priority of our experiments for multiple reasons. It does not correct images as well as the shutterless pipe, especially when the calibrations conditions stray away from the live conditions. It only uses two reference temperatures at a single ambient temperature, so it has good corrective power only for ambient temperatures close to the calibration temperature. Also, as seen in Figure 6, it leads to degraded performances in object detection. These results, as well as Lynred's best interests, gravitate us towards the shutterless pipe. However, it would be foolish not to try out some basic experiments for the shutter pipe. After all, it still runs faster than the shutterless method, and this is ultimately our goal, provided we can improve performances. Not only that, but for applications where the ambient temperature does not vary too much, such as indoor imaging, the shutter pipe can be very interesting.

First of all, we can examine how varying ambient calibration temperatures affects the detection performance down the line. We have images for ambient temperatures ranging from 10 to 50C, and for each ambient

temperature, we have a set of images with the camera pointing at a uniform object of temperature 10C (cold reference), and a set with the camera pointing at a uniform object of temperature 40C (hot reference). By varying the ambient temperature and feeding the corresponding calibration images for a hot and cold object, we get varying correction qualities. Figure 8 shows how the performance degrades when we stray away from an optimal temperature, in both directions. It is thus likely that the images were taken at an average temperature around 25C, and as the calibration temperature moves away from 25C, the correction quality decreases, leading to a decrease in the detection accuracy. The chart shows that we can get away with a temperature range between 20 and 30C for our current dataset.

When it comes to examining the effects of each algorithm in the correction pipe on the final detection, we do not have any statistically significant results that indicate that one of the algorithms plays a major role in driving performance. As seen earlier, the shutter pipe seems to perform decently well without any additional correction algorithms (other than the main NUC), and we even see a degradation in its performance when the algorithms are introduced all together. Upon closer examination, we realize that most algorithms do not significantly affect the performances. The baseline performance with piecewise tone-mapping only is somewhere around 66% mAP, and the introduction of algorithms does not significantly nudge it. The spatial denoising might be causing some degradation in performance, on the order of around 1%, but the main algorithm degrading performances seems to be the INL Correction, causing a drop of around 10 points. The baseline correction time per image is at 2ms, and adding spatial denoising brings it up to around 6ms, while adding destriping nudges it up to 12ms. The conclusion here hasn't changed much then, we'd probably be better off leaving the shutter pipe with tone-mapping only. Eliminating the need for other algorithms can reduce the total correction time from 19ms (with everything included), to around 2ms.

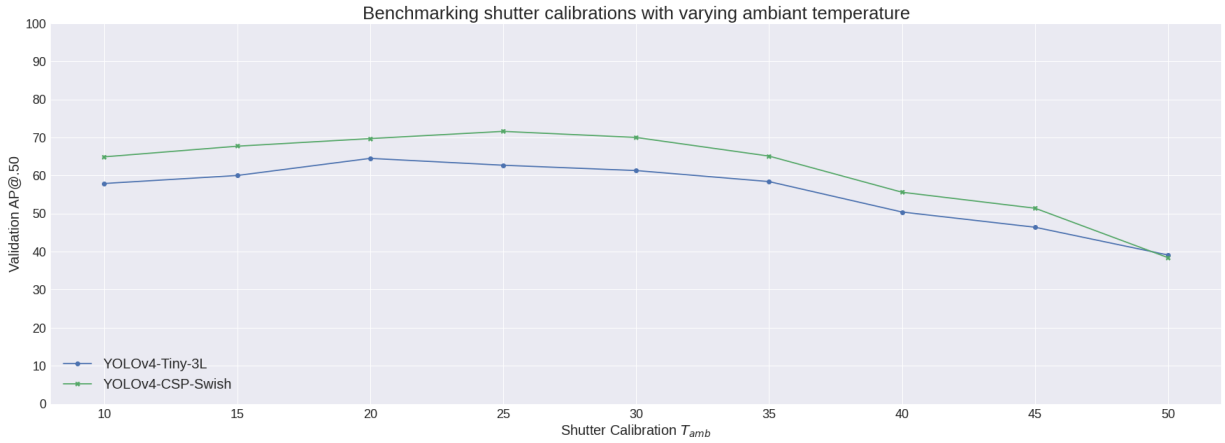

Figure 8: Benchmarking Shutter Calibration Temperatures

4.3.8 Benchmarking the Shutterless Pipe

We start out by testing out different calibration temperatures for the shutterless pipe. This experiment is a bit trickier for the shutterless pipe because we have an additional reference temperature. We vary the ambient temperatures of the main references from 10 to 50C, in 10C increments, and repeat the process for a third hot-reference at two different ambient temperatures, as shown in Figure 9. What we can extract from this is that the shutterless pipe is more robust to discrepancies between environmental and calibration conditions. Over a large range from 10C to 40C, we see a negligible variation in performance, which only starts to drop when we increase the ambient temperature to 50C, which is well outside the reasonable range for our conditions.

We can clearly see the differences in how the performance reacts to the calibration temperature variations in the shutterless pipe compared to the shutter pipe, but it would also help to take a look at the individual correction algorithms and how they differ, and more specifically, how each pipe's algorithms differ. A single image is passed through multiple processing pipes, for shutter and shutterless, with an NUC, piecewise tonemapping, and an additional correction algorithm, as shown in Figure 10. First off, we can quickly see how the visual quality from the shutterless pipe exceeds that of the shutter pipe, for all algorithms. Next, we can also understand the effect of spatial denoising (SDN): it removes noise but slightly blurs the image, diluting certain details and edges crucial to detecting objects. The effect of destriping is also clear, for both pipes, where it ends up removing column noise, which is especially striking when compared to other algorithms which do not deal with this column noise. The bad pixel replacement (BPR) algorithm can go by unnoticed, but in our example we have a few bad pixels on the rear side of the car, which show up as black dots, and which the BPR removes

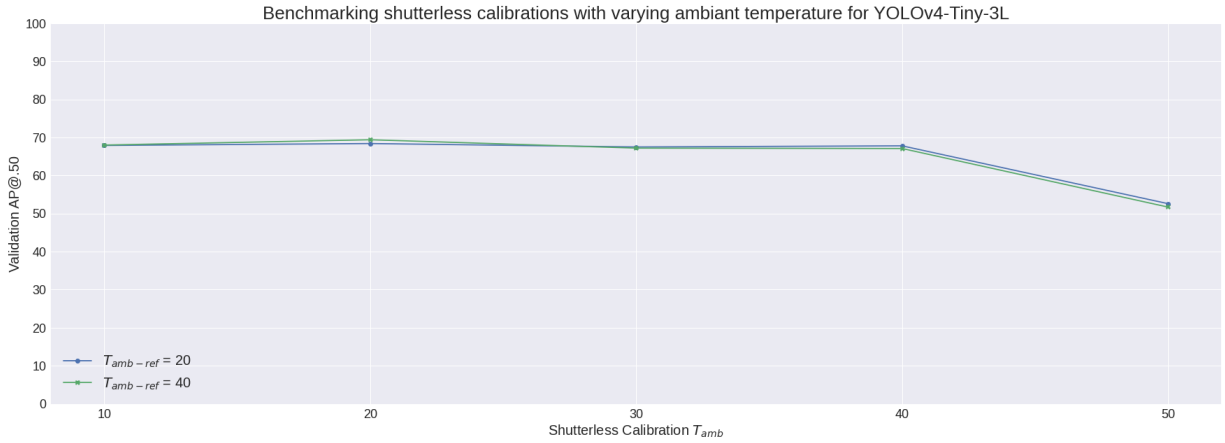

Figure 9: Benchmarking Shutterless Calibration Temperatures

quite well. The shutterless flare algorithm helps reduce the artifacts encountered as a result of a bright spot. In our case, the "bright" spot is in fact a "hot" spot, representing the heat radiating from the car engine, which shows up as a white spot on the infrared grayscale image, below the car. The flare algorithm helps reduce this effect. Finally, the shutter INL correction algorithm is the Integral Non-Linearity correction, which helps reduce noise due to the quantization of the signal (analog to digital) in the infrared sensor. However, in cases where this error is not very present, the INL algorithm can decrease the image quality, as we can see in this example, and as shown in the previous section by the reduction in performance it induces.

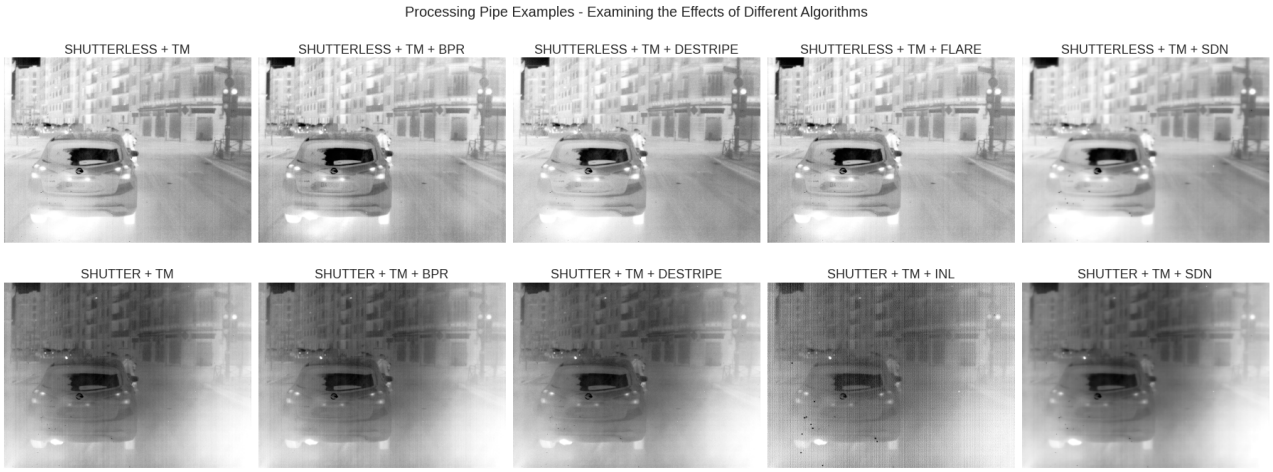

Figure 10: Samples showing the effects of individual shutter and shutterless correction algorithms

Having learned some lessons from the shutter trials, we realize that we do not need to test all listed algorithms. For example, the AGC (Automatic Gain Control) and Evolpel are only useful in live scenarios where the camera is running, and should have no effect on performances. Indeed, we find that the pipes produce identical images whether these algorithms are activated or not, so they are excluded from the experimentation pipeline. We are then left with Tone-Mapping (TM), Bad Pixel Replacement (BPR), Destriping (DESTRIP), Spatial Denoising (SDN), Temporal Denoising (TDN), and Flare (FLARE). These trials are shown in Figure 11. Each trial is repeated between 3 and 5 times, and the bars are sorted by increasing order of the overall testing accuracy. Although the differences in accuracy overall are not very noticeable when comparing to nearby bars, but the overall trend seems to indicate that some effects can be extracted from this chart. The first important effect is that of spatial denoising. The inclusion of spatial denoising seems to consistently degrade performances: the worst-performing trials all have spatial denoising included. This can be explained by the fact that spatial denoising reduces noise but makes the image slightly blurry. For human vision, this may translate into improved visual quality, but for an object detection model, this may translate into blurred edges and a loss of information, especially for pedestrians at a distance.

We might be able to say that BPR does not strongly affect performances, and this would make sense because it acts on a few pixel values, and that should not have much of an affect on a network, especially a convolutional one. Towards the right-end of the chart, it becomes a bit difficult to discern the individual effects of algorithms, but it can be said that temporal denoising and destriping might have a net positive effect on

detection, and warrant further investigation

The final strange effect observed in the chart is the extreme oscillation of the correction time red curve. It turns out that this is caused by the inclusion of the Temporal Denoising algorithm, and how the Lynred correction pipeline operates. The Lynred pipeline is conceived for correcting a live stream of images. In that scenario, the frames are all temporally correlated and it is easy to apply temporal denoising. In our scenario, we have a dataset saved on disk whose images are not all temporally related: images in a sequence are related but are not successive frames, and images from different sequences do not relate to them. This makes it that the correction pipe has a hard time applying its temporal correlation priors to non-related images, and it thus needs to be reinitialized after every correction in our offline training scenario. It turns out that the first image passed through a freshly initialized pipeline takes more time to correct than the second image. When temporal denoising trials were introduced, we started feeding the preceding frame image before every image from the dataset in order to apply a 2-frame temporal correction, and as it turns out, the correction pipe speeds up after the first introduced image, which is why we get a gain of around 10x when temporal denoising is introduced. For Lynred's purposes, this detection will mainly be performed in a live inference scenario, where the pipe is not reinitialized, where the temporal coherence between frames is preserved, and where the baseline correction time is rather uniform. As this was the first time we notice this effect, this experiment was deemed informative on the inner workings of Lynred's correction pipeline, and the differences we can expect between production and offline use. Finally, even for the correction times, we're interested in the relative differences between times as algorithms are introduced. In this context, the heaviest algorithms seem to be Flare and Destriping, and this should be taken into account when deciding whether or not to include them in the pipe.

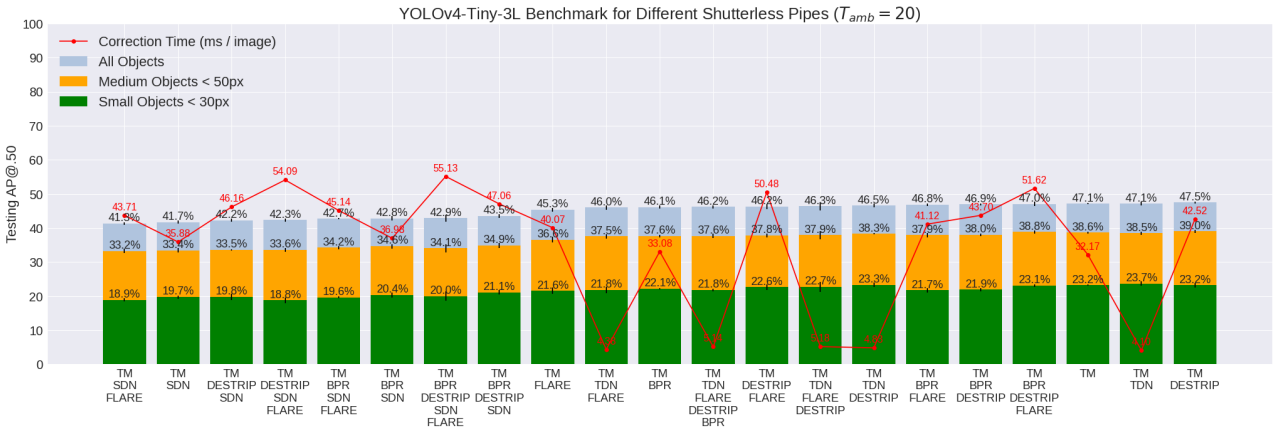

Figure 11: Benchmarking Shutterless Correction Algorithms

As seen above, we've determined that the destriping algorithm might be beneficial to the detection. How beneficial is it? Can we reduce its computational load? Can we improve its benefits? To answer these questions, we dive a bit deeper into the parameters of the destriping algorithm. As per Lynred's documentation, it seems to have three boolean parameters that determine whether to use smoothing, erosion, and score. The score parameter determines whether to add weighting to the gradient computation in the destriping algorithm. The erosion parameter controls whether we apply a horizontal morphological erosion on the obtained gradient map, and the smoothing parameter allows the algorithm to distinguish real vertical edges from stripe-type column noise. These boolean parameters are accompanied by parameters that control the window size of each of the applied methods. We start off by simply turning on or off these boolean parameters, and the results are shown in Figure 12. In these trials, we reintroduce evaluation on quantized versions of the models. What we see is that there isn't much of a difference in correction time between the different trials, and there isn't much of a difference in overall accuracy either. Besides the first trial which seems to have encountered some abnormal behavior (low mean and very high variance), the difference between the best and worst trial is only around 0.6%, which can be insignificant in the grand scheme of things. The effect on quantization isn't different from previously obtained conclusions: FP16 perform a bit worse that FP32, and INT8 performs a bit worse than FP16, without any surprises. Again, the slight performance degradation with FP16 is warranted due to the enormous speed gains. We might then conclude then out of all of these trials, we do not have a compelling reason to include destriping as a means of improving over a vanilla tone-mapping pipeline.

After eliminating spatial denoising from the experiments, we can try out combinations with temporal denoising as the main actor, in order to study its effects. In Figure 13, we do not see any compelling reason to use anything other than tone-mapping in this pipe. The introduction of any combination results in a slight degradation in performance and in correction time, naturally. These trials are an opportunity to show the generalization power of our algorithm, since the Lynred dataset is built using a single camera, and the noise

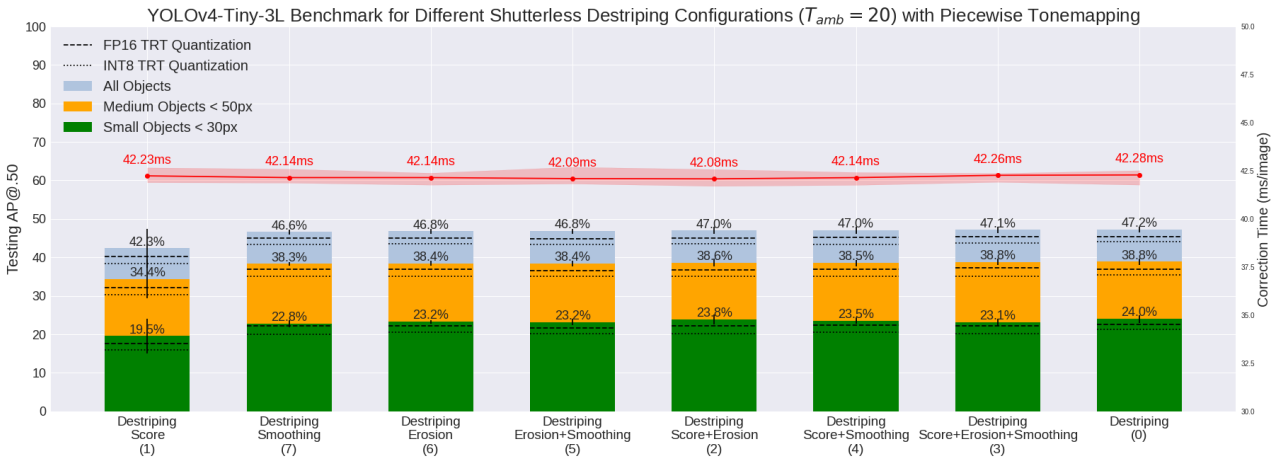

Figure 12: Benchmarking Shutterless Destriping Configurations

signature is different for different cameras. In addition to our test set and quantization, we also try out this model on the FLIR dataset, with the person class only. The results are impressive, and are shown in blue for each bar, with each blue line representing accuracy for small, medium, and all objects. Multiple conclusions can be extracted from these results: first, the Lynred VRU dataset is much more difficult than the FLIR dataset in terms of the person class, since accuracy on FLIR is almost 20 points above that on VRU. Second, our models then seem to generalize very well to other datasets, which isn't always the case in computer vision tasks. Even after training a model on raw images corrected with a minimal pipeline (NUC + Tone-Mapping), we still manage to obtain great results on a dataset whose infrared images are pre-corrected and taken in different conditions, different locations, and with different sensors.

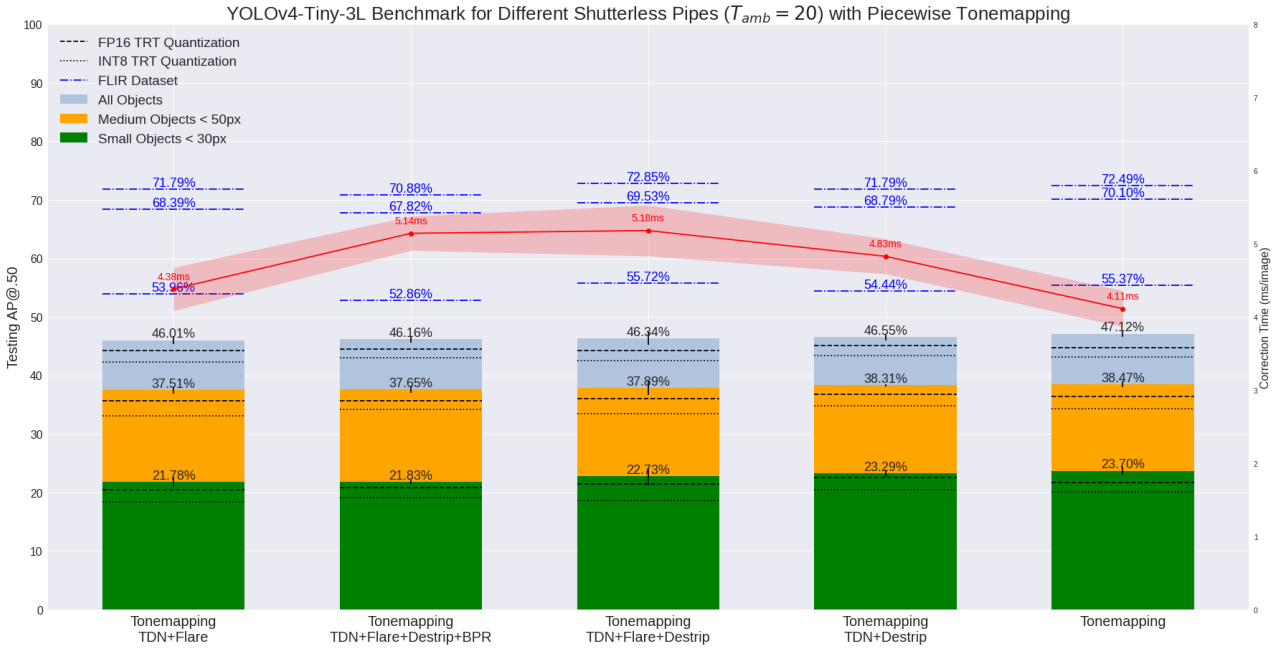

Figure 13: Benchmarking Shutterless Temporal Denoising Combinations for YOLOv4-Tiny-3L

Finally, Figure 14 shows similar experiments using the Swin-B + Cascade R-CNN model, to see whether models other than YOLO exhibit similar behavior. With the lack of statistical power, it can be said that these results are not very different from ones obtained before. Although we see a different trend with the tonemapping pipe scoring last and the TDN+Flare+Destrip pipe scoring first, the difference is not very significant for single runs.

Conclusions for the shutterless pipe are then as follows:

- The shutterless pipe performs better than the shutter pipe, by Lynred's experience and our experimentation.

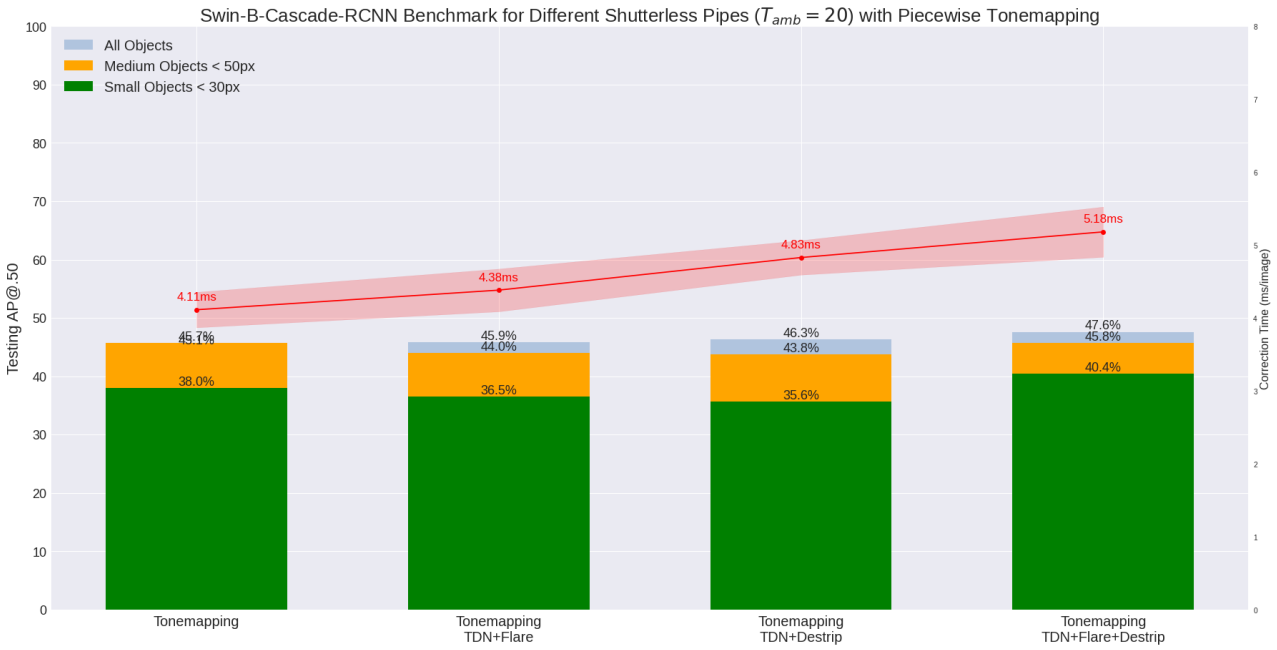

Figure 14: Benchmarking Shutterless Temporal Denoising Combinations for Swin-B + Cascade R-CNN

- Tone-mapping is required, and piecewise seems to be the best method.
- Spatial denoising degrades performances, and there is no use including it.
- Results are not very compelling in advocating for the use of other correction algorithms.
- In the long run, if these experiments are repeated a significant number of times, we might see a stronger benefit to the temporal denoising and destriping algorithms. While the temporal denoising algorithm is rather light, destriping is heavy and its contributions need to be significant for being worth including.
- The models we are using behave more or less the same way, even after quantization, and generalize very well to the FLIR dataset.

4.3.9 Thermal Noise Augmentation

To some extent, we've optimized the shutterless processing pipe, and early experiments with the shutter pipe indicate that it under-performs relative to the shutterless pipe. The shutterless pipe was shown to be more robust to variations in calibration temperature while the shutter pipe performs well only when the calibration conditions are closest to the capture conditions. Knowing that the shutter pipe has a lower computational load and lower correction time, can we improve the correction quality of the shutter pipe?

One way to do so is through data augmentation with thermal noise. The general idea is that if the shutter pipe behaves best with conditions close to the calibration, then by adding noisier images to the dataset, we can make the model more robust to differences in correction quality due to the shutter pipe performance. Thus, the idea is to add thermal noise to the images in our dataset before training the model, and see whether that improves the detection performance. There are two ways to go about this augmentation:

- The first is to inject additive thermal noise to the images corrected by a shutter pipe at a certain calibration temperature. This additive noise is created by taking two calibration images and subtracting them. As mentioned earlier, calibration are taken at a given ambient temperature T_{amb} and observe an object of uniform temperature T_{obj}. The creation of thermal noise is done by choosing two different T_{amb}, then choosing a single T_{obj}, giving us two images at $T_{amb1} - T_{obj}$ and $T_{amb2} - T_{obj}$. Subtracting these two images gives us a thermal noise of $T_{amb1} - T_{amb2}$. By generating random thermal noise and adding it to images, we would have created images augmented randomly with thermal noise at various temperatures.
- The second way of doing this augmentation is by varying the calibration temperature of the shutter pipe. For a given calibration, the shutter pipe will produce good quality images if their temperature is not far from the calibration temperature. By varying the shutter calibration from one image to another, we effectively create a variability in the correction quality, which should be akin to thermal noise (if the calibration temperature is too far from the image temperature, the images could be more noisy).

Classes	Infrared Val AP@.50	RGB Val AP@.50
truck	29.67	22.81
bus	59.38	57.64
person	79.24	70.72
motor	72.22	63.50
light	66.94	64.02
hydrant	38.21	58.90
bike	54.12	56.47
car	86.49	80.63
other vehicle	17.02	23.24
sign	59.32	58.32

Table 4: YOLOv4 performance on FLIR infrared and RGB validation sets

After having tried both of these augmentation techniques to augment the dataset 2x and 3x, the results were shown to be mostly identical with or without augmentation. So under the current paradigm, we were unable to improve the robustness of the model to thermal noise.

4.3.10 Multi-class detection with FLIR

A final experiment is one to further test the flexibility of our framework and robustness of our models on a different dataset. Although the person class was shown to be easier to learn in the FLIR dataset than in Lynred's VRU, FLIR has a total of 15 classes which can constitute a more challenging task overall. A YOLOv4 model is trained on the 10 most common FLIR classes, separately for IR and RGB images. The model's performance on FLIR's validation sets can be seen in Table 4, and an image-pair sample from the dataset is shown in Figure 15. As with the VRU dataset, we notice a similar pattern that justifies the use of infrared imaging in autonomous vehicle applications: the performance on IR images exceeds that on RGB images.

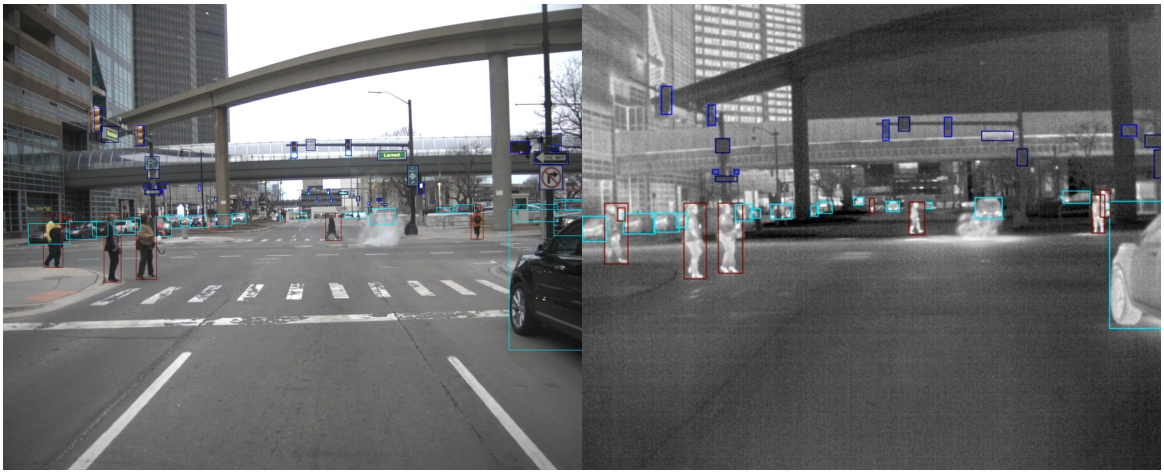

Figure 15: Sample Image-Pair from FLIR dataset with bounding boxes for RGB (left) and IR (right)

5 Conclusion

This work examined the intricacies of Lynred's raw infrared image correction pipeline in order to optimize its runtime and subsequent object detection performance. Although conclusions were not easy to obtain, several important highlights can nevertheless be extracted. The use of infrared images in autonomous driving applications, especially for pedestrian detection, is highly warranted, as infrared detection outperforms detection in the visible domain. The use of Lynred's correction pipeline is also crucial due to the inability of current models to extract information from highly noisy raw infrared images. Within the correction pipeline families, the shutterless thermal correction outperforms the shutter correction, as hypothesized by Lynred and confirmed by experiments. However, within these two pipes, one algorithm stands out as crucial to neural network training and inference: tone-mapping. Although Lynred initially thought of tone-mapping as simply a way to convey visual information to human eyes through computer monitors, it turns out that neural networks are accustomed to processing information in a very similar way as well, and Lynred's tone-mapping algorithm is a crucial step in their pipeline. Some algorithms in the pipeline are only useful in live scenarios with online correction and have no effect on detection, such as Evolpel and AGC, and other algorithms like spatial denoising seem to degrade performances. The remaining algorithms, notably Flare, Destriping, and Temporal Denoising, might be useful in the processing chain for optimal visual quality, but do not seem to bring great benefits to an object detection perform, and they slow down the correction. It then seems that the best option is to go for a lean and mean processing pipe with thermal noise correction and tone-mapping. These other algorithms might be shown to be significant in light of new experiments with a higher run count, for statistical significance.

As this thesis is still currently ongoing, there is more work to be done. Currently, we are focusing on optimizing performances for live inference on an Nvidia Jetson, and for this we need to go back and re-examine choices such as reading infrared images in three-channels. Single-channel inputs are much faster to process but lead to lower performances, and this may be improved by leveraging other datasets to create pretrained infrared detection models. As we near the end of this project, it becomes clearer and clearer that contrary to what the state-of-the-art might have you believe, speed is king.

References

[1] D. D. C. Shelton, "Writing in the first person for academic and research publication," Nov. 2015.

[2] I. P. C. Society, *Breaking the rules: "don't let good grammar spoil good writing" and passive/active voice*, IEEE, Ed., [Online; posted 22-Feburary-2017], Feb. 2017. [Online]. Available: https://procomm.ieee.org/breaking-the-rules-dont-let-good-grammar-spoil-good-writing-and-passiveactive-voice/.

[3] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[4] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, 2001, pp. I–I. DOI: 10.1109/CVPR.2001.990517.

[5] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, 2005, 886–893 vol. 1. DOI: 10.1109/CVPR.2005.177.

[6] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013. arXiv: 1311.2524. [Online]. Available: http://arxiv.org/abs/1311.2524.

[7] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015. arXiv: 1506.02640. [Online]. Available: http://arxiv.org/abs/1506.02640.

[8] Z. Cai and N. Vasconcelos, *Cascade r-cnn: Delving into high quality object detection*, 2017. arXiv: 1712.00726 [cs.CV].

[9] A. Bochkovskiy, C. Wang, and H. M. Liao, "Yolov4: Optimal speed and accuracy of object detection," *CoRR*, vol. abs/2004.10934, 2020. arXiv: 2004.10934. [Online]. Available: https://arxiv.org/abs/2004.10934.

[10] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[11] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale," *CoRR*, vol. abs/2010.11929, 2020. arXiv: 2010.11929. [Online]. Available: https://arxiv.org/abs/2010.11929.

[13] Z. Liu, H. Mao, C. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," *CoRR*, vol. abs/2201.03545, 2022. arXiv: 2201.03545. [Online]. Available: https://arxiv.org/abs/2201.03545.

[14] Z. Liu, Y. Lin, Y. Cao, *et al.*, "Swin transformer: Hierarchical vision transformer using shifted windows," *CoRR*, vol. abs/2103.14030, 2021. arXiv: 2103.14030. [Online]. Available: https://arxiv.org/abs/2103.14030.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems*, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[16] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015. arXiv: 1504.08083. [Online]. Available: http://arxiv.org/abs/1504.08083.

[17] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015. arXiv: 1506.01497. [Online]. Available: http://arxiv.org/abs/1506.01497.

[18] O. Russakovsky, J. Deng, H. Su, *et al.*, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. arXiv: 1409.0575. [Online]. Available: http://arxiv.org/abs/1409.0575.

[19] T. Lin, M. Maire, S. J. Belongie, *et al.*, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014. arXiv: 1405.0312. [Online]. Available: http://arxiv.org/abs/1405.0312.

[20] C. Wang, A. Bochkovskiy, and H. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," *CoRR*, vol. abs/2011.08036, 2020. arXiv: 2011.08036. [Online]. Available: https://arxiv.org/abs/2011.08036.

[21] C. Wang, I. Yeh, and H. M. Liao, "You only learn one representation: Unified network for multiple tasks," *CoRR*, vol. abs/2105.04206, 2021. arXiv: 2105.04206. [Online]. Available: https://arxiv.org/abs/2105.04206.

[22] A. Bochkovskiy (@alexeyab84), *New yolor (scaled-yolov4-based) is better than new convnext, swin, detr, yolox, pp-yolov2, yolov5, efficientdet... swin is so slow that it needs a logarithmic latency scale to fit on the chart...* Tweet, Jan. 2022. [Online]. Available: https://twitter.com/alexeyab84/status/1481982576818630662.

[23] C. Herrmann, M. Ruf, and J. Beyerer, "CNN-based thermal infrared person detection by domain adaptation," in *Autonomous Systems: Sensors, Vehicles, Security, and the Internet of Everything*, M. C. Dudzik and J. C. Ricklin, Eds., International Society for Optics and Photonics, vol. 10643, SPIE, 2018, pp. 38–43. DOI: 10.1117/12.2304400. [Online]. Available: https://doi.org/10.1117/12.2304400.

[24] TELEDYNE, *Can thermal imaging see through walls? and other common questions*, flir.com, Ed., [Online; posted 30-December-2020], Dec. 2020. [Online]. Available: https://www.flir.com/discover/rd-science/can-thermal-imaging-see-through-fog-and-rain/.

[25] S. Hwang, J. Park, N. Kim, Y. Choi, and I. S. Kweon, "Multispectral pedestrian detection: Benchmark dataset and baselines," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[26] TELEDYNE, *Teledyne flir free adas thermal dataset v2*, F. Conservator, Ed., [Online; posted 19-January-2019], Jan. 2019. [Online]. Available: https://adas-dataset-v2.flirconservator.com/#downloadguide.

[27] Q. Wang, Y. Chi, T. Shen, J. Song, Z. Zhang, and Y. Zhu, "Improving rgb-infrared object detection by reducing cross-modality redundancy," *Remote. Sens.*, vol. 14, p. 2020, 2022.

[28] J. Liu, X. Fan, Z. Huang, *et al.*, *Target-aware dual adversarial learning and a multi-scenario multi-modality benchmark to fuse infrared and visible for object detection*, 2022. DOI: 10.48550/ARXIV.2203.16220. [Online]. Available: https://arxiv.org/abs/2203.16220.

[29] J. Redmon, *Darknet: Open source neural networks in c*, http://pjreddie.com/darknet/, 2013.

[30] K. Chen, J. Wang, J. Pang, *et al.*, "MMDetection: Open mmlab detection toolbox and benchmark," *arXiv preprint arXiv:1906.07155*, 2019.

[31] N. Zmora, H. Wu, and J. Rodge, *Achieving fp32 accuracy for int8 inference using quantization aware training with nvidia tensorrt*, Nvidia, Ed., [Online; posted 20-July-2021], Jul. 2021. [Online]. Available: https://developer.nvidia.com/blog/achieving-fp32-accuracy-for-int8-inference-using-quantization-aware-training-with-tensorrt/.

[32] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14. DOI: 10.1109/ISSCC.2014.6757323.

Appendix

Planning and Task Chronology

Feburary: Getting Started

- Data analysis: analyzing dataset, individual images, labels
- Infrared processing: understanding main correction concepts and algorithms
- Object Detection State-of-the-Art: understanding model landscape and selecting potential candidates
- Object Detection Trials: training models on base pre-corrected datasets to get baseline performances and comparisons
- Object Detection Framework: start developing framework to group Darknet, MMDetection, Lynred SDK
- Image Correction SDK: start developing Python library for Lynred's correction SDK

March: Framework and Baselines

- Model Evaluation: develop metrics, outputs, and visualizations
- YOLO Model Benchmarks
- First Swin Experiments
- Image Correction SDK: integrate Lynred's SDK into framework
- Model Quantization: add TensorRT framework and quantization modules for Darknet models
- General Pipeline Benchmarks
- Shutterless Pipe Benchmarks

April: Experiment!

- Shutterless Binary Benchmarks
- Tonemapping Benchmarks
- More Specialized Shutterless Benchmarks
- Swin Transformer Benchmarks and Investigation

May: Refine and Dive In

- Refactor Detection Framework: allow for faster experimentation and more meaningful outputs
- Shutterless Destriping Benchmarks
- Shutterless Temporal Denoising
- FLIR Experiments
- Thermal Noise Augmentation Experiments version 1

June: Demo and Report

- Thermal Noise Augmentation Experiments version 2
- Robustness and Validation Experiments: test with different models and datasets
- Live Demo: prepare live inference on Jetson and optimize speed
- Final Report: finish up report for thesis

July: Optimize Product

- Report and Presentation: finalize deliverables for thesis
- Improve Shutter Pipe Performance
- Live Demo using Jetson, Quantization, Optimization

Abstract

As infrared imagery increasingly integrates the realm of deep learning-based computer vision, especially through applications such as autonomous driving, it becomes crucial to optimize performances and examine shortcomings of existing pipelines. Infrared imagery can help in low-visibility situations such as fog and low-light scenarios. In these cases, the visible images can be more strongly affected by the environment, while the infrared cameras provide a certain robustness to these external factors. This work studies the effect of different infrared processing pipelines on the performance of an object detection tackling pedestrians in an urban environment, similar to autonomous driving scenarios. It aims to optimize the processing pipeline for detection accuracy and detection speed, deviating from the current visual quality-focused pipes. The different experiments highlight a few conclusions. Detection on infrared images outperforms that on visible images for autonomous driving applications. The infrared correction pipeline is also crucial since the models cannot extract information from raw infrared images. Two thermal correction pipelines are studied, the shutter and the shutterless pipes. The shutter pipes works well when images are taken in an environment close to the calibration temperature, and is very fast. This pipe is a candidate for further optimizations and can be leveraged to run on embedded devices, especially for indoor applications where temperatures are stable. The shutterless pipe is very adaptable to the environment and can correct images outside of its calibration temperature range. As it runs slower, it is necessary to optimize it for speed and accuracy. Experiments show that most of the correction algorithms in the pipe are not very useful. Some, like spatial denoising, are detrimental to performance and needlessly add computational time, while others, like destriping and temporal denoising, show certain inconclusive benefits, but also add computational time. The tonemapping algorithm turned out to be crucial for processing information in neural networks and leveraging pretrained models. More experimentation may lead to exposing stronger benefits, but as it stands, the optimal trade-off for speed and accuracy is simply to use the shutterless pipe only with a tonemapping algorithm, for autonomous driving appliations with varied environments.

Résumé

Alors que l'imagerie infrarouge s'intègre de plus en plus dans le domaine de la vision par ordinateur basée sur l'apprentissage profond, notamment par le biais d'applications telles que la conduite autonome, il devient important d'optimiser les performances et d'examiner les faiblesses des pipelines existants. L'imagerie infrarouge peut être utile contre des facteurs qui diminuent la visibilité, comme le brouillard ou simplement l'obscurité. Dans ces cas, les images visibles peuvent être plus fortement affectées par l'environnement, tandis que les caméras infrarouges offrent une certaine robustesse à ces facteurs externes. Cette étude cherche à comprendre l'effet de différentes chaînes de traitement infrarouge sur la performance d'une détection de piétons dans un environnement urbain. Elle vise à optimiser la chaîne de traitement pour la précision et la vitesse de détection, plutôt que pour la qualité visuelle. Les différentes expériences mettent en évidence quelques conclusions. La détection sur les images infrarouges est plus performante que celle sur les images visibles pour les applications de conduite autonome. La chaîne de correction infrarouge est également primordiale puisque les modèles ne peuvent pas extraire l'information cachée dans les images infrarouges brutes. Deux chaînes de correction thermique sont étudiées, le shutter et le shutterless. Le shutter fonctionne bien lorsque les images sont prises dans un environnement proche de la température de calibration, et est très rapide. Ce pipe est un candidat pour des optimisations ultérieures et peut être exploité pour fonctionner sur des dispositifs embarqués, en particulier pour les applications intérieures où les températures sont stables. Le shutterless est très adaptable à l'environnement et peut corriger des images en dehors de sa plage de température de calibration. Comme il est plus lent, il est nécessaire de l'optimiser pour la vitesse et la précision. Les expériences montrent que la plupart des algorithmes de correction du shutterless ne sont pas très utiles. Certains, comme le débruitage spatial, nuisent aux performances et ajoutent inutilement du temps de calcul, tandis que d'autres, comme la réduction du bruit de colonne et le débruitage temporel, présentent certains avantages non concluants, mais ajoutent également du temps de calcul. L'algorithme de tonemapping s'est avéré crucial pour le traitement de l'information dans les réseaux de neurones et l'exploitation des modèles pré-entraînés. Une expérimentation plus poussée pourrait permettre de mettre en évidence des avantages plus importants, mais actuellement, le meilleur compromis entre vitesse et précision consiste simplement à utiliser le shutterless uniquement avec un algorithme de tonemapping, pour les applications de conduite autonome dans des environnements variés.