Emerging Information Science and Technology
Vol. 5, No. 1, (2024), pp. 36-45

Improving YOLO Object Detection Performance on Single-Board Computer using Virtual Machine

Muhamad Amirul Haq[1*], Le Nam Quoc Huy[2], Nuniek Fahriani[3]

[1,3]*Universitas Muhammadiyah Surabaya*
[2]*National Taiwan University of Science and Technology*
**Corresponding author : amirulhaq@ft.um-surabaya.ac.id*

Abstract

Single-board computers have gained popularity in the recent decade, largely due to the immense advancements in deep learning. Deep learning involves complex computational processes that are beyond the capabilities of regular microcontrollers, thus necessitating the use of single-board computers. However, single-board computers are primarily designed to operate efficiently in low-power environments. Therefore, optimization is crucial for running deep learning algorithms effectively on single-board computers. In this work, we explore the impact of utilizing the DeepStream framework to run deep learning algorithms, specifically the YOLO algorithm, on NVIDIA Jetson single-board computers. The DeepStream framework can be executed in virtual machines, notably Docker, to improve the performance and portability of the model. Additionally, deploying the Docker virtual machine from removable disks can further enhance its portability and even increase the algorithm's speed. The result of this study shows that real-time streaming of the YOLO algorithm can operate up to 8.5 times faster when deployed from a Docker virtual machine.

Keywords: *Deep learning, single-board computer, virtual machine, edge computing, optimization*

1. Introduction

Deep learning has become a critical component in the evolution of computing technology, notably impacting the functionality and efficiency of single-board computers (SBCs) [1]. These compact, energy-efficient devices have become increasingly vital in implementing intelligent systems, especially in applications like traffic surveillance and security. However, deploying deep learning algorithms on SBCs presents a unique set of challenges due to the devices' limited computational power and memory capacity. This situation calls for specialized optimization strategies to fully exploit the capabilities of SBCs without compromising performance [2], [3].

In the application of traffic surveillance, for example, real-time processing and analysis of video streams are essential for detecting accidents, managing congestion, and ensuring public safety [4], [5], [6]. Similarly, security applications rely on the swift recognition of potential threats and anomalies to prevent incidents [7]. Deep learning models, with their ability to learn from vast amounts of data and make accurate predictions, are ideal for these tasks. Yet, their complexity and the high volume of computations make them difficult to run effectively on SBCs, which are constrained by lower processing power and limited memory compared to standard computing resources.

The primary challenge in using SBCs for deep learning lies in balancing the need for computational efficiency with the devices' inherent limitations. SBCs, while capable of performing a wide range of computing tasks, are not inherently designed to handle the intensive demands of deep learning algorithms. This discrepancy can lead to issues such as slower processing speeds, reduced accuracy in model predictions, and the potential for overheating, which can affect system reliability and longevity.

Deploying and maintaining deep learning models on SBCs requires specialized expertise, particularly in fine-tuning deep learning algorithms to function within the the devices' limited resources. This optimization process involves several methods aimed at enhancing the operational efficiency of models while managing their demand on computational and memory resources. Some examples among these methods is model pruning, which involves the selective removal of less important neurons from a network. This technique effectively reduces the model's size and its computational complexity, thereby lightening the load on SBCs without significantly compromising the model's accuracy. Another example is quantization, which reduces the precision of the numerical parameters of a model from floating-point representations with longer decimal points to smaller datatypes. This approach can substantially decrease the model's memory footprint and speed up its computations, making deep learning models more adaptable to the constraints of SBCs. Knowledge distillation is also employed, where a simpler, more efficient "student" model learns to mimic the behavior of a larger, more complex "teacher" model. The student model maintains essential functionalities with much lower computational requirements. Optimization also extends to coding practices, which are crucial for minimizing the computational cost of SBCs. Techniques such as optimizing loops, utilizing efficient data structures, and reducing memory access times can considerably decrease the computational load. Additionally, exploiting the hardware acceleration capabilities of some SBCs, like GPUs or TPUs, can further enhance the performance of deep learning applications.

Optimizing deep learning models for use on SBCs involves several aforementioned techniques, include pruning (trimming unneeded parts of the model), quantization (reducing the precision of the calculations to speed them up), and knowledge distillation (teaching a smaller model to mimic a larger one). Additionally, writing code that makes good use of the computer's capabilities and using hardware acceleration where possible are also key. Despite the benefits of these methods, they require a deep understanding of both the models and the hardware, making it challenging without specialized knowledge in computer science or engineering. Deploying these optimized models on SBCs often involves dealing with various software and hardware requirements, which can make the process complicated. Different SBCs may have unique needs, such as specific operating systems or drivers. This complexity limits the ease of applying deep learning models in dynamic settings, such as for traffic surveillance or security systems, by affecting their scalability and portability.

The approach proposed here simplifies this process significantly. It uses NVIDIA Deepstream [8] and TensorRT [9], [10] for automatically optimizing the models, making them smaller and more efficient without losing accuracy. Then, by packaging these models into Docker containers [11], [12], the deployment becomes much easier across different SBCs. Docker containers help manage software dependencies and ensure that the models can run smoothly on various devices without needing extensive setup. This method effectively addresses compatibility and scalability issues, making it a practical solution for deploying deep learning applications on SBCs.

Experimental results from this research show that this approach not only makes deployment easier but also speeds up the running time of deep learning algorithms on multiple SBCs, including Jetson Nano, Jetson Xavier NX, Up Squared, and Up Xtreme. This improvement is crucial for real-time applications where quick processing and analysis are necessary for effectiveness.

In summary, this research provides a comprehensive strategy for deploying deep learning models on SBCs more efficiently. By automating the optimization process using Deepstream and TensorRT while simplifying deployment through Docker containers, it lowers the barrier to entry for using advanced deep learning applications across different devices. This streamlined approach enhances the feasibility of implementing intelligent

systems in areas requiring real-time processing capabilities, such as traffic surveillance and security, thereby expanding the potential applications of SBCs in the field of deep learning.

2. Method

In this methodology section, we detail the comprehensive approach used to evaluate the performance of the YOLOv3-tiny [13] model across various hardware platforms, focusing on its implementation in different runtime environments. This study was conducted to assess the effectiveness of deploying deep learning models, specifically for object detection and identification tasks, on SBCs and a standard Windows desktop. The goal was to explore optimizations that could facilitate the deployment and enhance the performance of deep learning models on devices with limited computational resources.

The methodology for conducting the experiments involved setting up each hardware platform with the necessary software and runtime environments. For the SBCs and the Windows desktop, this setup included the installation of Docker and the configuration of the Deepstream and TensorRT environment as needed. Three different application of the YOLOv3 (regular object detection, person identification, and pedestrian counter) model was then deployed and tested in the native runtime, regular Docker, and Deepstream Docker environments, with performance metrics collected for comparison.

Performance evaluation focused on measuring the inference speed and accuracy of the YOLOv3 tiny model across the different hardware platforms and runtime environments. The deployment from removable disks was assessed in terms of the ease of setup and any impact on performance metrics.

This comprehensive methodology allowed for a detailed analysis of the factors influencing the deployment and performance of deep learning models on various computing platforms, with a particular focus on optimizing for SBCs. The findings are expected to contribute valuable insights into the practical application of deep learning technologies in resource-constrained environments.

2.1. *You-Only-Look-Once* (YOLO) Algorithm

The deep learning model chosen for this study is YOLOv3 tiny, a lighter version of the YOLO (You Only Look Once) [14] model, known for its efficiency and speed in real-time object detection tasks. The architecture of the implemented YOLOv3-tiny is illustrated in Figure 1. We evaluated three distinct implementations of the YOLOv3 tiny model, each tailored to a specific application: Pascal VOC object detection [15], pedestrian counting, and person identification. These applications were selected to cover a broad spectrum of real-world scenarios where efficient real-time processing on edge devices is crucial.

For the object detection experiments, we utilized the Pascal VOC 2012 dataset, a benchmark in the field of computer vision. This dataset comprises 20 object classes, including people, animals, vehicles, and household objects, making it a comprehensive testbed for evaluating the effectiveness of object detection models. The diversity of classes and the challenging nature of the images, which often include objects in varied poses and occlusions, make the Pascal VOC 2012 dataset ideal for testing and benchmarking deep learning models. Its widespread use in the research community allows for direct comparison with other models and techniques, providing a clear measure of performance improvements.

The pedestrian counting implementation was designed to simulate a common application of surveillance and urban planning. For this purpose, an IP camera was set up overlooking a sidewalk in an urban area. The task involves detecting and counting pedestrians passing by in real time. This application tests the model's ability to accurately identify and track multiple individuals in dynamic, outdoor environments where lighting, weather, and crowd

density can vary significantly. The practical implications of this experiment extend to city planning, traffic management, and public safety, where accurate pedestrian counts are crucial for informed decision-making.

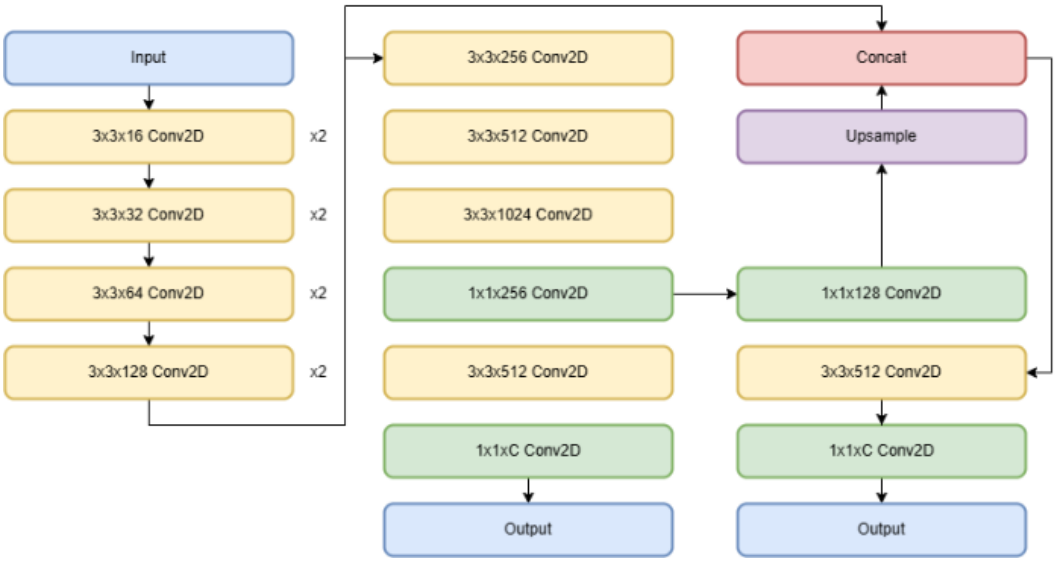

Figure 1. YOLOv3-tiny Network Architecture

For person identification, a webcam was installed in the hallway of an office building. This setup aims to evaluate the model's performance in recognizing and identifying individuals in a controlled indoor environment. The person identification task is critical for applications such as access control, attendance tracking, and security surveillance. Unlike the pedestrian counting setup, which focuses on detecting any pedestrian, person identification requires the model to discern specific individuals, often under varying lighting conditions and angles. This experiment highlights the model's capabilities in more personalized applications, where accuracy in identifying individuals is paramount.

These three implementations of the YOLOv3 tiny model—Pascal VOC object detection, pedestrian counting, and person identification—serve as comprehensive tests of the model's utility across a spectrum of real-world scenarios. From generic object detection with the diverse Pascal VOC 2012 dataset to the specialized tasks of counting pedestrians on a city sidewalk and identifying persons in an office building, these experiments underscore the model's adaptability and performance in addressing different challenges faced in the deployment of deep learning models on single-board computers.

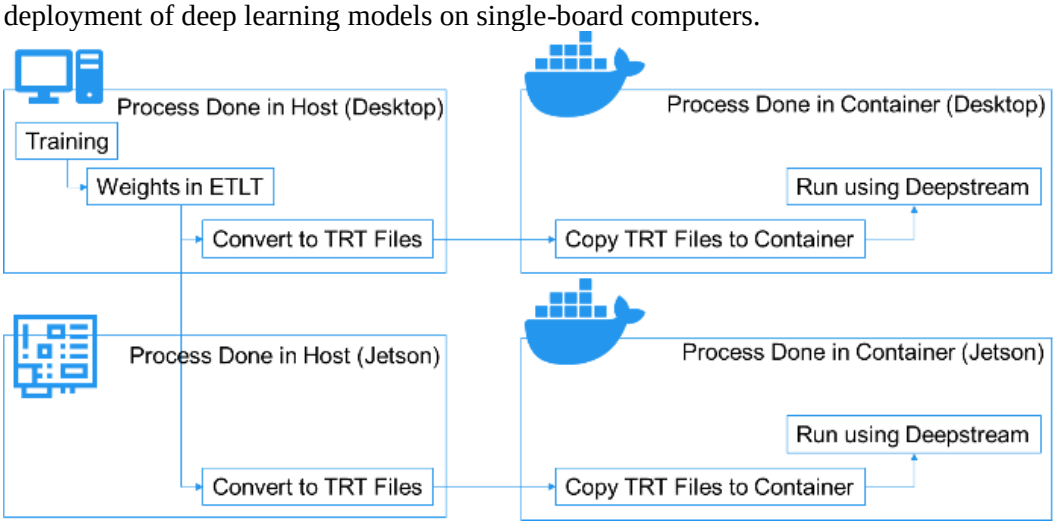

Figure 2.Docker allows cross-platform deployment. Allowing models to be trained only once

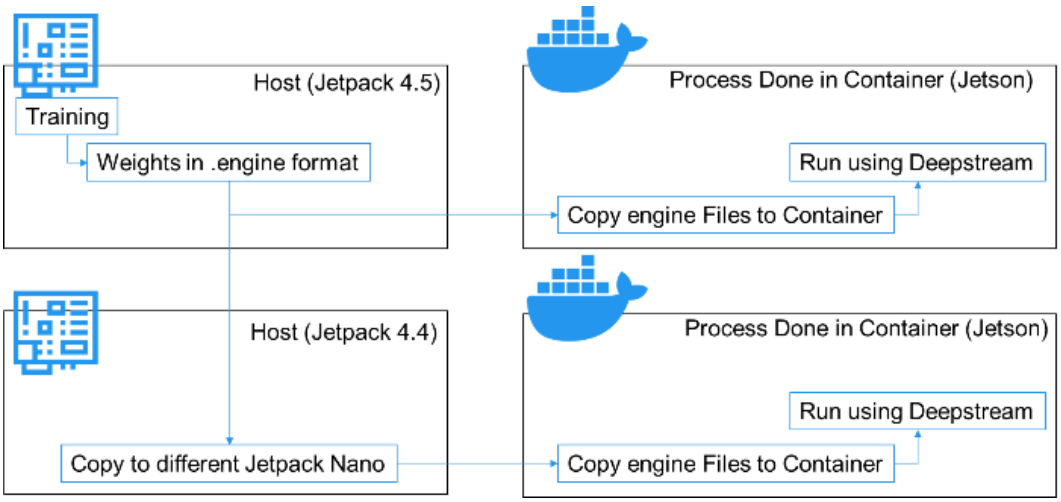

Figure 3. Models can be trained in any devices with different OS version and then deployed via Docker

2.2. Docker Virtual Environment

Docker plays a pivotal role in this research by providing a platform for containerization, a technique that packages an application along with its dependencies and libraries in a container. This containerization ensures that the application runs consistently across different computing environments, as illustrated in Figure 2 and 3. In the context of deploying deep learning models on single-board computers (SBCs), Docker offers several advantages that are instrumental to the research:

2.2.1. Consistency Across Environments

By encapsulating the deep learning environment and its dependencies within a Docker container, the research ensures that the deep learning models can be deployed consistently across different SBCs. This consistency is crucial for comparative analysis and benchmarking, as it eliminates the variability in performance that can arise from differences in the underlying software environment.

2.2.2. Portability

Docker containers can be easily transferred and deployed across different systems, enhancing the portability of the deep learning models. This portability is particularly beneficial in research settings where models need to be tested on various hardware platforms to assess their performance and scalability.

2.2.3. Isolation

Docker provides an isolated environment for each container, which minimizes conflicts between different software versions or dependencies. This isolation is important for research, as it allows for simultaneous testing of multiple configurations or models without interference, ensuring the integrity of the experimental results.

2.2.4. Efficiency

Docker's lightweight nature and its ability to share resources across containers make it an efficient choice for deploying deep learning models on SBCs, which often have limited resources. This efficiency supports more effective utilization of the available computational power, enhancing the performance of the deep learning models.

2.3. Deepstream and TensorRT Deep Learning Framework

The combination of NVIDIA DeepStream and TensorRT is integral to optimizing deep learning models for real-time applications in the research. DeepStream provides a streaming

analytics toolkit designed for AI-based multi-sensor processing, video, and image understanding. TensorRT is an SDK for high-performance deep learning inference, which includes a deep learning inference optimizer and runtime that delivers low latency and high throughput for deep learning applications. Together, they play several critical roles in the research:

2.3.1. Performance Optimization

Deepstream and TensorRT optimize deep learning models for inference on NVIDIA hardware, including Jetson SBCs. TensorRT accelerates deep learning inference through optimizations such as parametrization, while Deepstream allows for efficient processing and analysis of video streams.

2.3.2. Scalability

The integration of Deepstream with TensorRT enables the research to scale deep learning models to process video streams from multiple sources simultaneously, a key requirement for applications like traffic surveillance and security monitoring.

2.3.3. Real-time Processing

Deepstream facilitates the development of applications that require real-time video analytics, which is essential for the research's focus on real-time object detection and identification tasks. The combination with TensorRT ensures that these tasks can be performed with minimal latency, meeting the requirements for real-time performance on SBCs.

2.3.4. Ease of Development

DeepStream provides pre-built models and components that simplify the development of complex video analytics applications. This ease of development allows the research to focus on the application-specific aspects of the models, such as tuning for object detection or person identification, rather than the intricacies of video processing and inference optimization.

By leveraging Docker for deployment and NVIDIA DeepStream and TensorRT for performance optimization, the research addresses the challenges of deploying and running deep learning models on resource-constrained SBCs, enabling efficient and scalable real-time video analytics applications.

2.4. Removable Disk Deployment

An additional layer of experimentation involved exploring the feasibility of deploying the YOLOv3 tiny model from removable disks. This approach aimed to investigate the portability and ease of deployment across different devices, an essential factor for applications requiring flexibility and quick setup in various operational contexts.

2.5. Platform Setups

Table 1. Platform architecture and OS comparison

Machine	Arch	OS
Desktop	X86	Windows 11
Xavier NX	ARM	Jetpack OS
Up Xtreme	X86	Ubuntu OS
Up Squared	X86	Ubuntu OS
Nano	ARM	Jetpack OS

The experiments are conducted on five different platforms. The SBCs used for these experiments includes two Ubuntu x86-based SBCs (Up Xtreme and Up Squared), two

ARMs based SBCs (Jetson Xavier NX and the Jetson Nano) both running NVIDIA custom linux distro Jetpack OS. Additionally, a Windows desktop was employed as a baseline for performance comparison. The choice of these devices reflects a range of computational capabilities, from the relatively powerful Windows desktop to the more resource-constrained SBCs. Meanwhile, all models are implemented using Pytorch [16], [17] deep learning framework for fair comparison.

3. Results and Discussion

The Results and Discussion section of this study meticulously evaluates the performance of deep learning models, specifically the YOLO algorithm, across various computing platforms and configurations. This evaluation is crucial for understanding the effectiveness of our proposed solution, leveraging Docker and NVIDIA's DeepStream and TensorRT, in optimizing these models for single-board computers (SBCs). Our analysis is structured around three pivotal aspects: model accuracy, inference time, and resource consumption, each represented by detailed experimental data.

Table 2. YOLO accuracy on Pascal VOC Dataset

Models	***Accuracy (mAP)***
Pre-trained	*51.8 %*
120 epochs unpruned	*78.3 %*
300 epochs unpruned	*79.7 %*
300 epochs pruned (TRT)	*79.8 %*

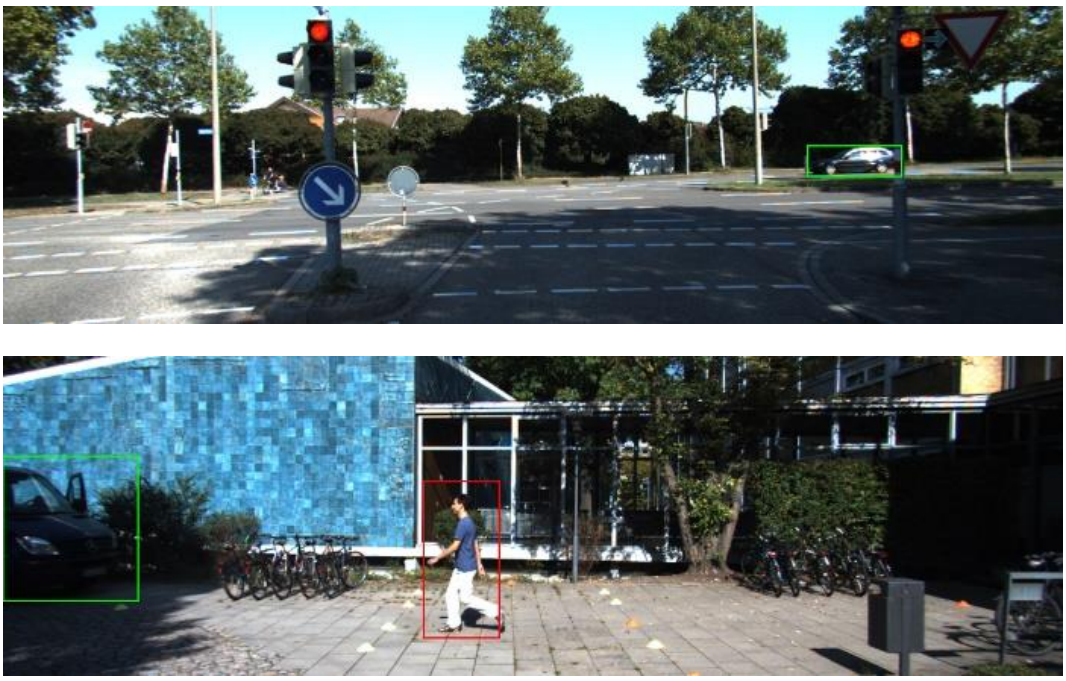

Figure 4. Visualization of object detection deployment on KITTI [18] **dataset. The model is trained on Pascal VOC 2012.**

3.1. Model Accuracy and Test Performance

Initially, we focus on the accuracy of the YOLO model, utilizing the Pascal VOC 2012 dataset. As illustrated in Table 2 and visualized in Figure 4, the model demonstrates a notable improvement in accuracy with extended training, moving from a pre-trained accuracy of 51.8% to 79.7% after 300 epochs. Interestingly, a slight accuracy increase is observed when applying pruning and TensorRT optimizations, pushing the accuracy to

79.8%. This increment, although modest, highlights the efficacy of model optimization techniques in not only maintaining but potentially enhancing model accuracy even after significant reduction in model complexity. However, there is also an inherent limitation of YOLO models, which is its inability to accurately detect overlapping objects, as shown in Figure 5. Nevertheless, the model also able to be run successfully on the two other tasks, the person identification and pedestrian counting, as demonstrated in Figure 6 and 7, respectively.

Table 3. Inference Time Comparison Between Docker and Native Runtime on Various Platforms

Machine	on Host	on Docker Container
Desktop	2 ms	3.3 ms
Xavier NX	41 ms	26 ms
Up Xtreme	32 ms	33.5 ms
Up Squared*	-	185 ms
Nano	619 ms	72.75 ms

** Up Squared cannot run Pytorch natively*

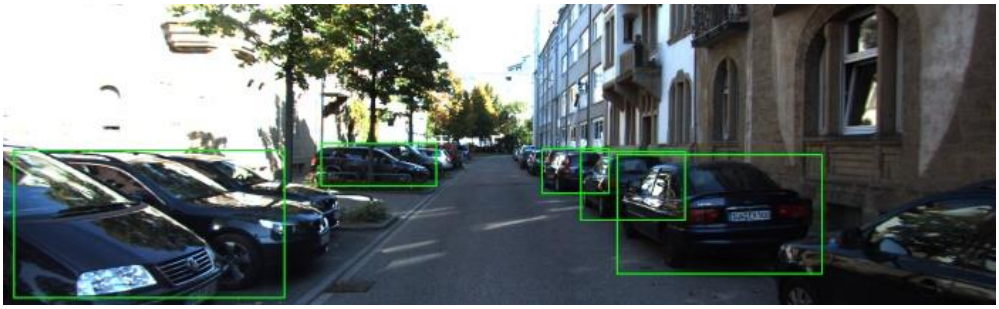

Figure 5. Several misdetection on the YOLOv3-tiny object detection algorithm. The model is especially prone to miss overlapping objects.

3.2. Inference Time Comparison Between Docker and Native Runtime

Table 3 presents a comparison of inference times between models running natively and those within a Docker container equipped with DeepStream. The data reveals a mixed impact of containerization on inference speed. While native execution is unsurprisingly faster on the desktop, running YOLO within a Docker container on the Xavier NX and Nano shows a dramatic decrease in inference time, showcasing the effectiveness of our Docker-based deployment in optimizing runtime performance on SBCs. Notably, the Up Squared SBC, which cannot run Pytorch natively due to hardware limitations, becomes capable of executing the model within a Docker environment, albeit with higher inference times, underscoring the versatility of our approach.

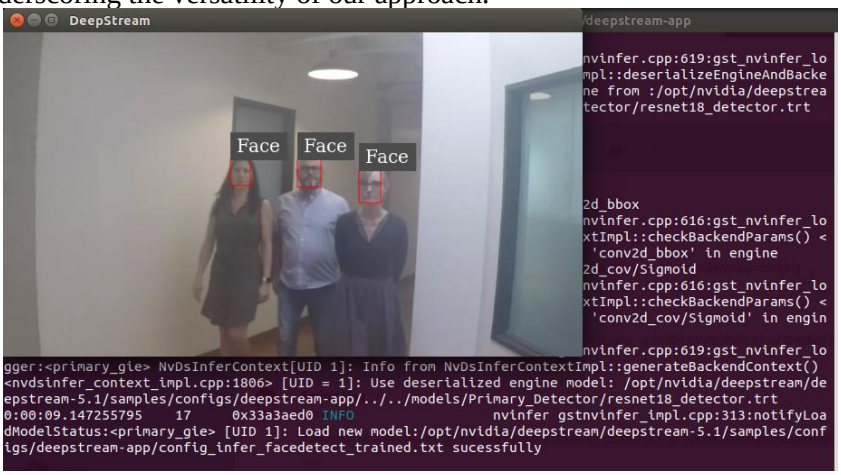

Figure 6. YOLO for Person Identification

3.3. Deepstream and TensorRT Deep Learning Framework

Table 4 explores the impact of running multiple Docker containers on resource usage within a desktop environment. The experiment aims to assess the scalability of our solution when deployed in scenarios requiring multiple instances of the model. The results indicate a marginal increase in CPU usage and a slightly more pronounced rise in RAM usage with the addition of more containers. However, the overall system resources remain largely unaffected, suggesting that our Docker-based deployment method is not only efficient but also scalable, allowing for multiple models to run concurrently without significant resource drain.

In summary, the experimental results substantiate the proposed method's effectiveness in enhancing the deployment and performance of deep learning models on SBCs. By achieving a balance between accuracy, runtime efficiency, and resource utilization, this research contributes meaningful insights into the practical application of deep learning in real-time, resource-constrained environments with the help of DeepStream framework [19], [20]. Further discussion will delve into the implications of these findings for the deployment of AI applications on edge devices and identify potential areas for future investigation.

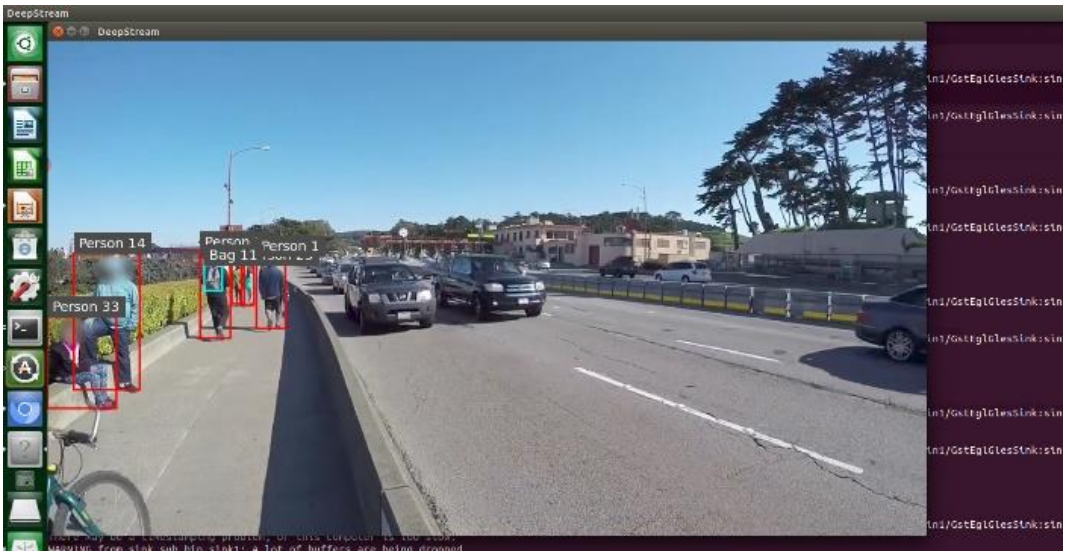

Figure 7. YOLO for Pedestrian Counter

4. Conclusion

Our experiments across various hardware platforms, including the Jetson Xavier NX and Nano, as well as Ubuntu-based x86 SBCs like the Up Xtreme and Up Squared, demonstrate a substantial improvement in the speed and efficiency of deep learning algorithms that is run on virtual machine. The deployment from removable disks further underscores the versatility and practicality of our approach, offering a new dimension of portability and ease of setup that is particularly valuable in dynamic application environments. The core challenge lies in the resource constraints of SBCs, which necessitate innovative approaches to model optimization and deployment. Our exploration reveals that leveraging Docker, combined with the computational optimizations offered by NVIDIA Deepstream and TensorRT, provides a compelling solution to these challenges. Docker significantly simplifies the deployment process across different SBCs by ensuring consistency and portability, while also offering efficient use of limited resources through its containerization technology. Meanwhile, Deepstream and TensorRT have proven essential in reducing the number of parameters of deep learning models on SBCs, enabling real-time processing capabilities that are crucial for the applications studied. This research not only contributes to the academic understanding of deep learning deployment on SBCs but also provides a

practical framework that can be readily applied across different domains requiring real-time data processing and analysis. Future work will explore further optimizations and the potential integration of emerging technologies to enhance the scalability and applicability of deep learning models in real-world scenarios, ensuring that the benefits of AI can be more broadly realized in a variety of settings.

References

[1] J. Chen and X. Ran, "Deep Learning With Edge Computing: A Review," *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, Aug. 2019, doi: 10.1109/JPROC.2019.2921977.
[2] M. Satyanarayanan, "The Emergence of Edge Computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017, doi: 10.1109/MC.2017.9.
[3] M. G. Brahmam and V. A. R, "VMMISD: An Efficient Load Balancing Model for Virtual Machine Migrations via Fused Metaheuristics With Iterative Security Measures and Deep Learning Optimizations," *IEEE Access*, vol. 12, pp. 39351–39374, 2024, doi: 10.1109/ACCESS.2024.3373465.
[4] U. Drolia, K. Guo, and P. Narasimhan, "Precog: prefetching for image recognition applications at the edge," in *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, in SEC '17. New York, NY, USA: Association for Computing Machinery, Oct. 2017, pp. 1–13. doi: 10.1145/3132211.3134456.
[5] M. A. Haq, S.-J. Ruan, and J.-H. Chen, "Detecting Obstacle in 3D Space using Monocular Camera," in *2022 IEEE 4th Global Conference on Life Sciences and Technologies (LifeTech)*, Mar. 2022, pp. 431–432. doi: 10.1109/LifeTech53646.2022.9754879.
[6] M. A. Haq, S.-J. Ruan, M.-E. Shao, Q. M. U. Haq, P.-J. Liang, and D.-Q. Gao, "One Stage Monocular 3D Object Detection Utilizing Discrete Depth and Orientation Representation," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 11, pp. 21630–21640, Nov. 2022, doi: 10.1109/TITS.2022.3175198.
[7] Z. Lv, D. Chen, B. Cao, H. Song, and H. Lv, "Secure Deep Learning in Defense in Deep-Learning-as-a-Service Computing Systems in Digital Twins," *IEEE Transactions on Computers*, vol. 73, no. 3, pp. 656–668, Mar. 2024, doi: 10.1109/TC.2021.3077687.
[8] Md. I. Uddin, Md. S. Alamgir, Md. M. Rahman, M. S. Bhuiyan, and M. A. Moral, "AI Traffic Control System Based on Deepstream and IoT Using NVIDIA Jetson Nano," in *2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST)*, Jan. 2021, pp. 115–119. doi: 10.1109/ICREST51555.2021.9331256.
[9] E. Jeong, J. Kim, S. Tan, J. Lee, and S. Ha, "Deep Learning Inference Parallelization on Heterogeneous Processors With TensorRT," *IEEE Embedded Systems Letters*, vol. 14, no. 1, pp. 15–18, Mar. 2022, doi: 10.1109/LES.2021.3087707.
[10] G. Jocher *et al.*, "ultralytics/yolov5: v6.1 - TensorRT, TensorFlow Edge TPU and OpenVINO Export and Inference," *Zenodo*, Feb. 2022, doi: 10.5281/zenodo.6222936.
[11] I. Miell and A. Sayers, *Docker in Practice, Second Edition*. Simon and Schuster, 2019.
[12] J. Nickoloff and S. Kuenzli, *Docker in Action, Second Edition*. Simon and Schuster, 2019.
[13] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement." arXiv, Apr. 08, 2018. doi: 10.48550/arXiv.1804.02767.
[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.
[15] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *Int J Comput Vis*, vol. 111, no. 1, pp. 98–136, Jan. 2015, doi: 10.1007/s11263-014-0733-5.
[16] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf
[17] "PyTorch | SpringerLink." Accessed: Apr. 01, 2024. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-030-57077-4_10
[18] "Vision meets robotics: The KITTI dataset - A Geiger, P Lenz, C Stiller, R Urtasun, 2013." Accessed: Apr. 01, 2024. [Online]. Available: https://journals.sagepub.com/doi/full/10.1177/0278364913491297
[19] V. Pednekar, N. Shettigar, and S. Tawhare, "Enhancing Security Surveillance Through Business Intelligence with NVIDIA DeepStream," in *Data Science and Emerging Technologies*, Y. Bee Wah, D. Al-Jumeily OBE, and M. W. Berry, Eds., Singapore: Springer Nature, 2024, pp. 91–104. doi: 10.1007/978-981-97-0293-0_7.
[20] Y. Anvarjon, S. Park, and J. Kim, "Design and Implementation of Data Concentrator Unit supported with Multiple Synchronized Cameras for Object-Detection," in *2023 IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)*, Oct. 2023, pp. 1–2. doi: 10.1109/ICCE-Asia59966.2023.10326437.