

---

# BENCHMARKING DEEP LEARNING MODELS ON NVIDIA JETSON NANO FOR REAL-TIME SYSTEMS: AN EMPIRICAL INVESTIGATION

---

**Tushar Prasanna Swaminathan**  
 Electrical and Computer Engineering  
 Lakehead University  
 Thunder Bay, Canada  
 tswamina@lakeheadu.ca

**Christopher Silver**  
 Electrical and Computer Engineering  
 Lakehead University  
 Thunder Bay, Canada  
 crsilver@lakeheadu.ca

**Thangarajah Akilan**  
 Department of Software Engineering  
 Lakehead University  
 Thunder Bay, ON P7B5E1  
 takilan@lakeheadu.ca

June 26, 2024

## ABSTRACT

The proliferation of complex deep learning (DL) models has revolutionized various applications, including computer vision-based solutions, prompting their integration into real-time systems. However, the resource-intensive nature of these models poses challenges for deployment on low-computational power and low-memory devices, like embedded and edge devices. This work empirically investigates the optimization of such complex DL models to analyze their functionality on an embedded device, particularly on the NVIDIA Jetson Nano. It evaluates the effectiveness of the optimized models in terms of their inference speed for image classification and video action detection. The experimental results reveal that, on average, optimized models exhibit a 16.11% speed improvement over their non-optimized counterparts. This not only emphasizes the critical need to consider hardware constraints and environmental sustainability in model development and deployment but also underscores the pivotal role of model optimization in enabling the widespread deployment of AI-assisted technologies on resource-constrained computational systems. It also serves as proof that prioritizing hardware-specific model optimization leads to efficient and scalable solutions that substantially decrease energy consumption and carbon footprint.

**Keywords** Deep learning, edge devices, optimization, NVIDIA Jetson Nano, TensorRT.

## 1 Introduction

In the rapidly evolving landscape of DL, the size of models capable of handling complex tasks, viz. image classification, action recognition, object detection, and semantic segmentation has increased dramatically. This growth is driven by the abundant intricate information that these models need to extract and perceive from the input data. Among them, ResNet [1], VGG [2], Inception [3], and U-Net [4] are examples of commonly utilized architectures. Meanwhile, the embedded and edge devices play a crucial role in performing real-time computation within operational environments. Deploying DL models on these devices eliminates the necessity for centralized cloud and data centers, leading to quicker and more reliable processing. Moreover, not everyone, be it an individual or an organization, can afford expensive high-performance computers (HPC) to deploy large DL models for practical applications. This problem is not only relevant to large DL models for large organizations' use cases, but there is also a notable surge in integrating

DL technologies into mobile applications [5]. This is especially relevant use case, as there will be a projected 6.38 billion smartphone users in the world [6]. Therefore, there is a need for optimizing existing models to run efficiently on devices with low computation and memory capacities.

This work extends beyond merely addressing the high computational demands on resource-constrained devices; it enables organizations to explore various modalities and combinations of deep learning models, thereby catalyzing breakthroughs in their respective fields that would otherwise be impractical on edge devices. For instance, Silver and Akilan [7] demonstrated that integrating supervised and unsupervised models into a meta-model classifier yields superior performance compared to using each model individually. This approach is particularly beneficial for scenarios where collecting a comprehensive dataset is challenging. However, such implementations on edge devices remain impractical without model optimization to reduce complexity. In this direction, this work pragmatically investigates DL model optimization for computer vision applications in-depth and conducts empirical analysis using the NVIDIA Jetson Nano.

The rest of this paper is organized as follows: Section 2 reviews related works and outlines the problem formulation. Section 3 elaborates on methodology and experimental setup. Section 4 discusses the experimental findings. Finally, Section 5 concludes the paper with future research directions.

## 2 Background

The literature indicates that while DL models are typically designed to produce high performance in their intended applications, they often struggle with real-time inference speed, a critical requirement for many practical applications. Consequently, models require meticulous optimization when deployed on edge devices, as the optimization strategies differ significantly from those used for specific hardware configurations. Without such tailored optimization, the models risk malfunctioning or failing entirely on the new hardware platform. This lack of compatibility can constrain the deployment options and flexibility of deep learning (DL) solutions developed by engineers. Conversely, unoptimized models fail to efficiently utilize available hardware resources, such as central processing units (CPUs), graphics processing units (GPUs), or specialized accelerators like tensor processing units (TPUs). This inefficiency can result in elevated computational costs and extended inference times. Hence, inefficient resource usage can lead to higher infrastructure costs, especially in cloud-based environments, where resources are typically billed based on usage [8].

### 2.1 Cloud vs Edge Devices

Numerous industrial applications are increasingly adopting Infrastructure as a Service (IaaS) provided by cloud vendors. However, the use of cloud computing services introduces several challenges, notably latency, which critically impacts real-time applications. For example, the frames captured by an autonomous vehicle’s camera must be processed immediately to prevent collisions with obstacles. Transmitting this video data over cloud networks often requires extensive bandwidth and memory, consequently leading to delays. Additionally, scalability issues arise, causing network access bottlenecks when data is simultaneously transmitted to the cloud from multiple sources. Privacy also remains a significant concern when transmitting sensitive data to the cloud. In contrast, edge devices in operational environments, such as the aforementioned autonomous vehicle, can process data in real-time. They only transfer essential information, such as the results of object detection in a video stream, to the cloud server when necessary, optimizing bandwidth usage and reducing latency. Images and other critical user information are processed on the edge device without transferring them to the cloud, which protects from data being tapped or stolen [9].

### 2.2 DL Model Optimization

It is widely recognized that as deep learning (DL) architectures grow in size and complexity, they necessitate the use of highly computationally intensive devices, which consequently increases their carbon footprint. Model optimization, however, offers an effective solution to mitigate these environmental impacts. Studies demonstrate that models with higher resource requirements can be optimized and run on lower computation devices, resulting in a smaller carbon footprint [10]. To address this, some researchers are concentrating on efficient hyperparameter optimization (HPO) that is suitable for deep learning (DL) models. Malik *et al.* [11] propose the use of less computationally demanding proxy tasks during the training phase of DL models to conserve resources. Additionally, inefficient models tend to consume more power, thereby increasing energy consumption. In contrast, optimized models that demand fewer computational resources can significantly reduce the environmental impact associated with AI and DL deployments. Supporting such advancements, certain hardware platforms offer specialized features, such as tensor cores in GPUs, to boost model performance and integrate concepts like TinyML [12–14].

There is also a growing demand for the development of new machine learning (ML) and neural network (NN) libraries that are optimized for use with field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs). For instance, Safaei *et al.* [15] introduce a ground-up approach to implement an extreme learning machine (ELM) classifier tailored to a system-on-a-chip (SoC) FPGA to gain computational acceleration, resulting in higher efficiency than general-purpose processors. Hence, Wang *et al.* [16] introduce a hardware-friendly model optimization technique that is highly efficient on low-storage devices. Similarly, Sertic *et al.* [14] performed hardware-specific optimization of various object detection models to deploy them on Intel Neural Compute Stick 2 and Google Coral Edge TPU. Consequently, this paper benchmarks various optimized deep learning (DL) models on a NVIDIA Jetson Nano edge device, focusing on two computer vision tasks: image classification and human action recognition.

### 3 The Methodology

#### 3.1 Environmental Setup

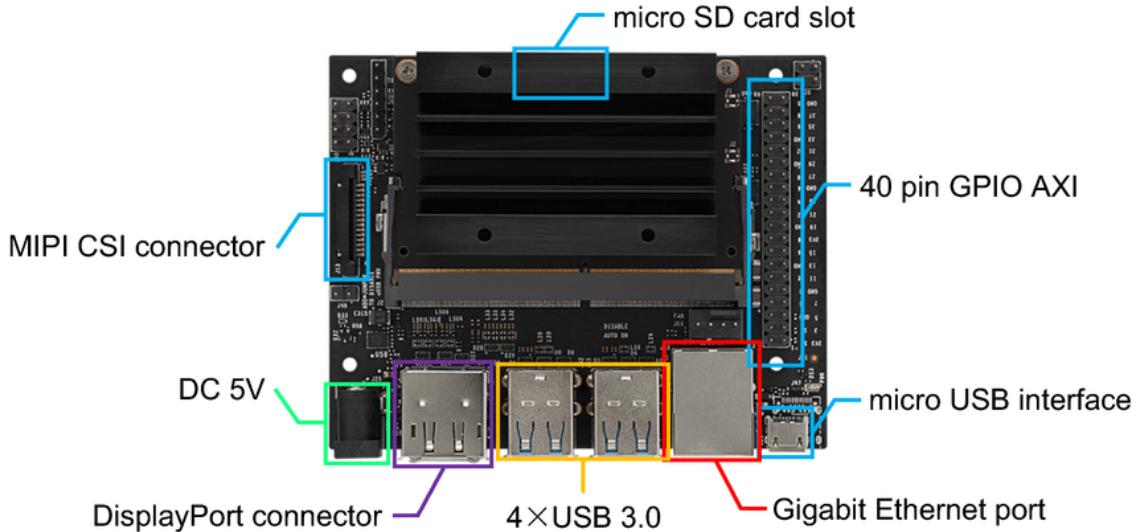


Figure 1: Layout of the NVIDIA Jetson Nano Developer Kit showcasing its key components and connectivity options [17].

This section elaborates on the hardware and software tools used in this study. The experimental study utilizes the NVIDIA Jetson Nano shown in Fig. 1, a compact and computationally powerful platform capable of parallel inference of multiple models. This platform is appropriate for various applications, including image classification, object detection, segmentation, tracking, and speech-to-text processing [18]. It consists of a CPU Quad-core ARM Cortex-A57 MP Core processor and GPU NVIDIA Maxwell architecture with 128 NVIDIA compute unified device architecture (CUDA) cores. The NVIDIA Maxwell architecture is suitable for deep learning processing tasks. The CUDA cores are the processing units responsible for performing computations on NVIDIA GPUs. The CUDA Toolkit provided by NVIDIA is a development environment for creating high-performance GPU-accelerated applications on the Jetson Nano [19].

The models are developed and tested using the PyTorch deep learning framework, which offers robust compatibility with TensorRT and facilitates straightforward conversion to ONNX format. TensorRT, an SDK by NVIDIA, is designed for real-time model inference, optimizing deep learning models to achieve low latency and high throughput. For model optimization and running inference with TensorRT, we use the NVIDIA (L4T R32.7.1) PyTorch Container for Jetson JetPack 4.6.1 [20]. This Docker container helps manage all compatible dependencies efficiently in a single environment, thereby preserving the integrity of external files.

#### 3.2 Experimental Setup

The experiments are divided into two categories: image classification and human action recognition. For image classification, we utilize several pre-trained deep learning (DL) models from the native PyTorch library [21], including AlexNet, VGG, ResNet, SqueezeNet, DenseNet, ShuffleNet-V2, MobileNet-V2, and ResNet-V2. These models,

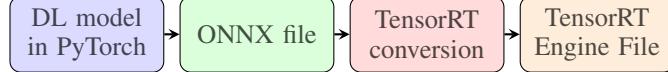


Figure 2: The PyTorch deep learning model optimization process for a NVIDIA Jetson Nano Edge Device using TensorRT.

known for their extensive use in various computer vision applications, have a high number of trainable parameters and large sizes, making them resource-intensive but ideal for benchmarking purposes in this study. All models are adapted for a binary image classification task. For human action recognition, three custom models are developed: a 3D-CNN, an Autoencoder (AE), and a DenseNet, all designed to process video input streams. Training is conducted using the UCF50 dataset [22], which includes fifty different action categories. However, this study narrows its focus to a 4-way action classification to simplify the analysis.

### 3.3 The Pipeline

Each model is optimized using TensorRT as mentioned above using the pipeline shown in Fig.2.

The conversion of a PyTorch deep learning model into a TensorRT engine comprises three main steps. The first step involves exporting the PyTorch model to an ONNX (Open Neural Network Exchange) file using the `torch.onnx.export()` function. This function traces the model’s execution with a specified input tensor and exports the traced model to ONNX format. By default, the exported ONNX graph possesses fixed input sizes for all dimensions. To accommodate input tensors with dynamic batch sizes, it is crucial to set the first dimension as dynamic using the `dynamic_axes` parameter of `torch.onnx.export()`. Exporting models to ONNX with dynamic shapes is essential for optimizing GPU memory usage and enhancing flexibility in processing varying input batch sizes. From the ONNX file, the TensorRT engine is built using the TensorRT Python API. This involves several stages, including creating a network definition, importing the model through the ONNX parser, and building the TensorRT engine with a builder.

The inference process with the TensorRT engine comprises six sub-steps (cf. Fig. 3): (i) creation of an inference execution context, (ii) memory allocation for input and output on the CUDA device, (iii) input data is transferred from the host into the input memory allocated on the CUDA device, (iv) TensorRT engine performs inference using the asynchronous execute API, (v) the output is transferred back into the host memory, and (vi) the stream used for data transfers and inference execution is synchronized to ensure the completion of all operations. This workflow allows for efficient deployment of PyTorch DL models on TensorRT-enabled platforms, leveraging the performance optimization capabilities of TensorRT for accelerated inference [23]. The TensorRT optimized file, i.e., `.trt` is generated through various techniques, like weight and activation precision calibration, layer and tensor fusion, kernel auto-tuning, dynamic tensor memory, and multi-stream execution methods for efficient inference by taking advantage of the NVIDIA Jetson GPU and its CUDA cores.

Table 1: Performance comparison of base models (Pre-Opt) and their optimized counterparts (Post-Opt.) on a NVIDIA Jetson Nano before and after optimization. The Inference Time Speedup is Computed Against the Respective Pre-Opt Models.

Model	Pre-Opt. Time (s)	Post-Opt. Time (s)	Pre/Post Time Ratio	Trainable Params	FLOPS	Inference Time Speed up
Image Classification Models						
AlexNet	0.6638	0.1184	2.09	61,751,008	1,475,805,888	5.62×
VGG	2.5904	0.4285	6.05	143,667,240	19,590,954,654	6.05×
ResNet	1.0911	0.2233	4.88	25,557,032	3,883,453,952	4.88×
SqueezeNet	0.7966	0.1663	4.79	1,235,496	780,414,144	4.72×
DenseNet	1.0132	0.3682	2.16	7,978,856	2,845,996,288	2.72×
ShuffleNet V2	4.7121	0.3463	13.61	2,278,604	298,632,608	13.6×
MobileNet V2	5.0379	0.3003	16.77	3,504,872	300,356,480	16.7×
ResNet V2	2.0964	0.2600	8.06	25,028,904	3,866,082,816	8.07×
Human Action Recognition Models						
3D-CNN	0.3431	0.0928	3.70	20,333,956	116,551,168	3.7×
AutoEncoder (AE)	0.7435	0.242	30.71	332,807	22,434,944	3.07×
DenseNet	3.1619	0.3718	8.50	109,386	27,681,924	8.5×

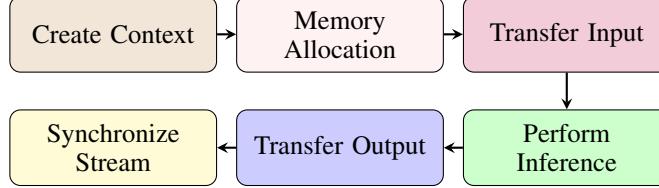


Figure 3: Inference process of TensorRT engine on NVIDIA Jetson Nano.

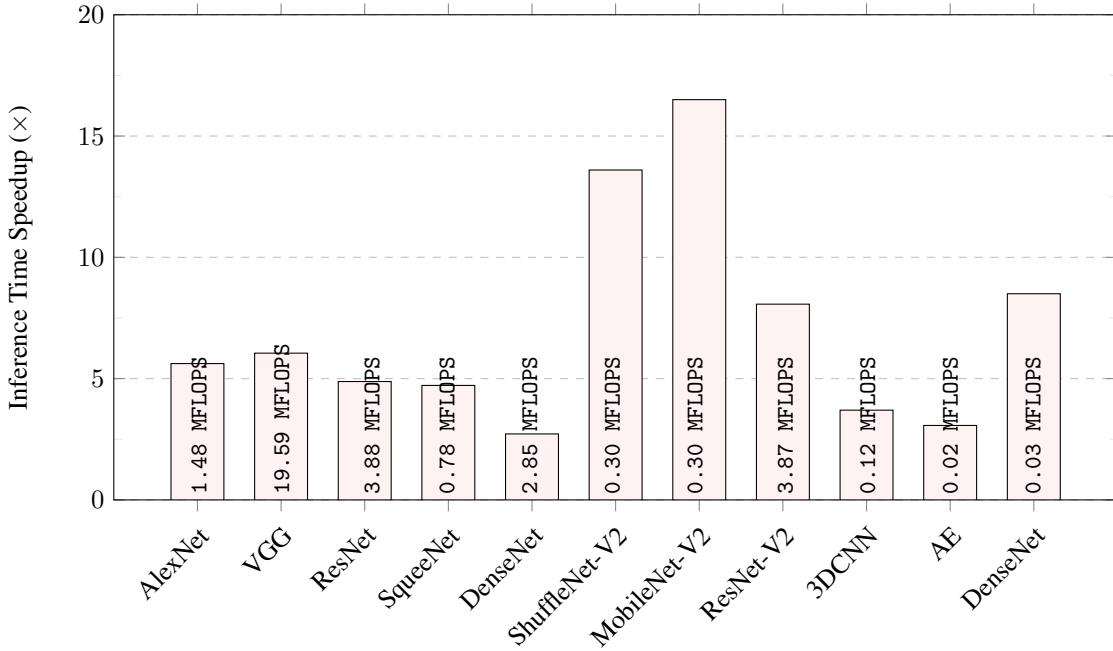


Figure 4: Inference time speedup of the optimized models on NVIDIA Jetson Nano compared to their non-optimized baseline counterparts.

## 4 Results and Discussions

To test the optimized models, various parameters are taken into consideration, like floating point operations per second (FLOPS), and the inference time of the models before and after the optimization, noted hereafter as *Pre-Opt* time and *Post-Opt* time, respectively. Table 1 summarizes the models' details, including FLOPS and the number of trainable parameters as they are critical factors that affect inference time. FLOPS serves as a crucial metric for assessing the computational complexity of deep learning models. Typically, higher FLOPS implies a greater number of computations during inference, potentially leading to longer inference times.

For instance, models, like VGG with approximately 19 billion FLOPS exhibit a considerable decrease in inference time, achieving only a  $6.05\times$  speedup after optimization. Upon analyzing the numerical values in the table, a discernible trend emerges, wherein models with lower FLOPS tend to experience a more substantial decrease in inference time following optimization efforts. For example, Shuffle-V2 and MobileNet-V2 showcase speed up of  $13.6\times$  and  $16.7\times$ , respectively, aligning with their low FLOPS values.

Despite the overarching trend, there exist exceptions within the dataset. Notably, ResNet-V2 possesses a significant number of FLOPS, yet its decrease in inference time is sped up at  $8.07\times$  compared to other models with comparable FLOPS. This variation could stem from the unique architectural characteristics of ResNet and the effectiveness of optimization techniques applied during the model's development. Thus, it is clear that several factors contribute to the variations in the relationship between FLOPS and inference time, viz. optimization techniques, and network architecture, including the number of parameters, depth, and connectivity patterns that influence the model's computational complexity and subsequently the inference time.

According to Fig. 4, there is a general trend where complex or large models show a minor reduction in inference time. Nevertheless, optimization positively affects the inference time across the models. However, the video-based action recognition models, specifically DenseNet and AutoEncoder (AE), do not conform to this trend. This divergence may be attributed to their architectural designs and the inefficiency of layer optimization, stemming from their status as custom-created rather than pre-trained models, which typically undergo extensive optimization.

## 5 Conclusion and Future Work

Considering the environmental impact of hardware devices, sustainable growth in AI deployment involves optimizing models rather than solely relying on hardware upgrades. By reducing computational complexity and resource requirements, optimized models contribute to minimizing energy consumption and carbon footprint, aligning with the goals of sustainability and environmental conservation. The results of this empirical study demonstrate the significant impact of optimization techniques on various DL-driven computer vision models. On average the optimized models are  $7.011\times$  faster than their non-optimized baseline counterparts. By leveraging tools, like TensorRT, the inference time of complex models is substantially reduced, making them more suitable for time-sensitive applications. The relationship between FLOPS and inference time highlights the importance of optimization in managing computational complexity and improving performance. Despite variations among models, the overarching trend showcases the effectiveness of optimization in decreasing inference time. While this work primarily focuses on optimization using TensorRT, future research can investigate other optimization techniques, such as quantization-aware training (QAT), and network pruning to further improve model efficiency and scalability. In conclusion, optimizing deep learning (DL) models for edge and embedded devices, such as the NVIDIA Jetson Nano, is essential for enabling efficient and scalable AI solutions. These optimization techniques address the challenges associated with cloud-based model deployment and are pivotal for fostering sustainable growth in AI and enhancing edge computing capabilities in microcontrollers and single-board computers. For further details and access to the source code of this work, the reader can refer to the project repository.

## References

- [1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015.
- [2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” *arXiv preprint arXiv:1409.4842*, 2014.
- [3] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *International Conference on Learning Representations*, 2015.
- [4] R. Azad, E. Khodapanah Aghdam, A. Rauland, Y. Jia, A. Haddadi Avval, A. Bozorgpour, S. Karimijafarbigloo, J. P. Cohen, E. Adeli, and D. Merhof, “Medical image segmentation review: The success of u-net,” *arXiv preprint arXiv:2211.14830*, 2022.
- [5] J. Liu and D. Ellis, “Editorial: Eating in the age of smartphones: The good, the bad, and the neutral,” *Frontiers in Psychology*, vol. 12, p. 796899, 2021.
- [6] A. Turner, “Number of mobile phone & smartphone users.” BankMyCell, March 2024.
- [7] C. Silver and T. Akilan, “A novel approach for fall detection using thermal imaging and a stacking ensemble of autoencoder and 3d-cnn models,” in *2023 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pp. 71–76, 2023.
- [8] J. Ariza, M. Jimeno, R. Villanueva-Polanco, and J. Capacho, “Provisioning computational resources for cloud-based e-learning platforms using deep learning techniques,” *IEEE Access*, vol. 9, pp. 89798–89811, 2021.
- [9] J. Chen and X. Ran, “Deep learning with edge computing: A review,” *Proceedings of the IEEE*, vol. 107, no. 8, pp. 1655–1674, 2019.
- [10] U. Gupta, Y. G. Kim, S. Lee, J. Tse, H.-H. S. Lee, G.-Y. Wei, D. Brooks, and C.-J. Wu, “Chasing carbon: The elusive environmental footprint of computing,” in *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 854–867, 2021.
- [11] N. Mallik, E. Bergman, C. Hvarfner, D. Stoll, M. Janowski, M. Lindauer, L. Nardi, and F. Hutter, “Priorband: Practical hyperparameter optimization in the age of deep learning,” *Advances in Neural Information Processing Systems*, vol. 36, 2024.

- [12] F. Oliveira, D. G. Costa, F. Assis, and I. Silva, “Internet of intelligent things: A convergence of embedded systems, edge computing and machine learning,” *Internet of Things*, p. 101153, 2024.
- [13] E. Jeong, J. Kim, and S. Ha, “Tensorrt-based framework and optimization methodology for deep learning inference on jetson boards,” *ACM Trans. Embed. Comput. Syst.*, vol. 21, no. 5, 2022.
- [14] P. Sertic, A. Alahmar, T. Akilan, M. Javorac, and Y. Gupta, “Intelligent real-time face-mask detection system with hardware acceleration for covid-19 mitigation,” in *Healthcare*, vol. 10, p. 873, MDPI, 2022.
- [15] A. Safaei, Q. J. Wu, T. Akilan, and Y. Yang, “System-on-a-chip (soc)-based hardware acceleration for an online sequential extreme learning machine (os-elm),” *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 38, no. 11, pp. 2127–2138, 2018.
- [16] P. Wang, W. Chen, X. He, Q. Chen, Q. Liu, and J. Cheng, “Optimization-based post-training quantization with bit-split and stitching,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 2, pp. 2119–2135, 2023.
- [17] Z. Qinghe, X. Tian, M. Yang, and H. Su, “Clmip: cross-layer manifold invariance based pruning method of deep convolutional neural network for real-time road type recognition,” *Multidimensional Systems and Signal Processing*, vol. 32, 01 2021.
- [18] S. Mittal, “A survey on optimized implementation of deep learning models on the nvidia jetson platform,” *Journal of Systems Architecture*, vol. 97, pp. 428–442, 2019.
- [19] “Nvidia jetson nano developer kit.” <https://developer.nvidia.com/embedded/jetson-nano-developer-kit>. Accessed: Sept 15, 2023.
- [20] “Nvidia l4t pytorch.” <https://catalog.ngc.nvidia.com/orgs/nvidia/containers/l4t-pytorch>. Accessed: Sept 19, 2023.
- [21] PyTorch, “PyTorch Vision Models,” 2023. Accessed: Oct 10th, 2023.
- [22] K. K. Reddy and M. Shah, “Recognizing 50 human action categories of web videos,” *Machine vision and applications*, vol. 24, no. 5, p. 971–981, 2012.
- [23] Y. Zhou and K. Yang, “Exploring tensorrt to improve real-time inference for deep learning,” pp. 2011–2018, 2022.