Comparative Analysis of TensorRT Inference Speeds Across Various Neural Network Block Architectures

Hwa-Jong Park
Intekplus
hjpark@intekplus.com

Abstract

This paper presents a systematic investigation into the inference speed of neural network block architectures across varying configurations of normalization layers, activation functions, and input resolutions. Four widely adopted block structures—VGG [16], ResNet [7], MobileNet [9], and ConvNeXt [12]—are evaluated under combinations of Batch Normalization (BN) [10] and Layer Normalization (LN) [2], ReLU [13] and GELU [8], and three input resolutions (64×64, 256×256, 1024×1024) using TensorRT optimizations in C++. The results show that MobileNet [9] consistently achieves the fastest inference speeds, being approximately 3x faster than ConvNeXt [12] at 64×64 resolution and 1.24x faster than VGG [16] and ResNet [7] across all tested configurations. These findings provide actionable insights for selecting and optimizing neural network architectures in latency-sensitive environments. The source code is available at: github.com/Hwa-Jong/PyTorch-to-TensorRT-Block-Comparison.

1 Introduction

The rapid advancements in Artificial Intelligence (AI) have driven its adoption across various industrial domains, such as robotics, autonomous driving, and manufacturing [1, 4, 6, 11, 14, 17]. In these fields, AI enables automation, enhances operational precision, and facilitates real-time decision-making, thereby transforming traditional industrial processes.

To utilize the strengths of both Python and C++, one widely adopted workflow involves training AI models in Python, converting them to ONNX, and deploying them with TensorRT in C++. This hybrid approach leverages Python's rich ecosystem for model development and C++'s high-speed capabilities, making it particularly suited for real-time applications [3].

In this paper, we systematically investigate the impact of neural network block architectures on inference speed when deploying models using TensorRT. Specifically, we evaluate the performance of four widely-used block structures—VGG Block [16], ResNet Block [7], MobileNet Block [9], and ConvNeXt Block [12]—trained in Python and optimized for C++ deployment through TensorRT. Through a comprehensive evaluation across varying configurations, we aim to provide practical insights for optimizing inference performance and deployment efficiency, enabling advanced and high-performance AI solutions for dynamic industrial applications.

2 Background

2.1 ONNX

Open Neural Network Exchange (ONNX) is an open-source standard for representing machine learning models. It facilitates seamless interoperability between various deep learning frameworks, such as PyTorch, TensorFlow, and Keras. ONNX provides a standardized format for seamless model export and import, eliminating the need for extensive adjustments during deployment.

By leveraging ONNX, models trained in Python-based frameworks can be converted into a format that is compatible with high-performance inference environments like C++ and TensorRT. This compatibility is particularly valuable in industrial applications, where flexibility and efficiency are critical. ONNX also supports a wide range of operators, making it suitable for diverse architectures.

Key benefits of ONNX include:

- **Interoperability**: Facilitates smooth transitions between training and deployment environments.
- **Optimization**: Enables integration with tools like ONNX Runtime to achieve faster inference.
- **Scalability**: Supports a wide range of architectures, from lightweight to highly complex models.

2.2 TensorRT

NVIDIA TensorRT is a high-performance deep learning inference library designed to optimize and accelerate neural network models for deployment in production environments. It is specifically developed to reduce latency and increase throughput for neural network inference on NVIDIA GPUs. TensorRT achieves this through advanced optimization techniques, such as kernel fusion, precision calibration (e.g., FP32 to FP16 or INT8), and layer fusion.

TensorRT is well-suited for real-time and resource-constrained applications. By integrating with ONNX, TensorRT enables developers to import pre-trained models from popular frameworks and optimize them into highly efficient inference engines for deployment.

Key features of TensorRT include:

- **Performance Optimization**: Utilizes advanced techniques, such as kernel fusion and dynamic tensor memory allocation, to maximize throughput.
- **Precision Tuning**: Supports FP32, FP16, and INT8 precision, balancing speed and accuracy based on deployment requirements.
- **Flexibility**: Adapts to various deployment requirements, from lightweight architectures to complex networks.

Together, ONNX and TensorRT form a robust pipeline that bridges the gap between Python-based model training and high-performance deployment in C++ environments, enabling scalable AI solutions for industrial applications.

3 Method

In this section, we describe the neural network block architectures evaluated in this study, including VGG [16], ResNet [7], MobileNet [9], and ConvNeXt [12]. Each architecture represents a unique approach to designing deep learning models, with varying trade-offs in complexity, performance, and efficiency. An overview of the architecture is depicted in Figure 1.

3.1 Overall Architectures

The network begins with an input stem, which transforms the input image into feature representations. The input stem consists of a 7×7 convolution, followed by Batch Normalization (BN) [10], ReLU activation [13], and MaxPooling. Each stage of the network consists of multiple blocks, and the detailed structures of these blocks—including VGG, ResNet, MobileNet, and ConvNeXt—are illustrated in Figure 2. Finally, the output head comprises a Global Average Pooling (GAP) layer and a Linear layer, which maps the extracted features to the desired number of output classes.

To ensure experimental fairness, the number of activation layers was standardized across all architectures at each stage. This approach facilitates a consistent basis for comparison while accommodating potential variations in the number of blocks per stage, which arise from differences in architectural design.

3.2 VGG

VGG [16] is a foundational architecture known for its simplicity, relying on stacked convolutional layers with fixed kernel sizes. This straightforward design enables strong performance in image classification tasks. The detailed structure of the VGG Block is illustrated in Figure 2(a).

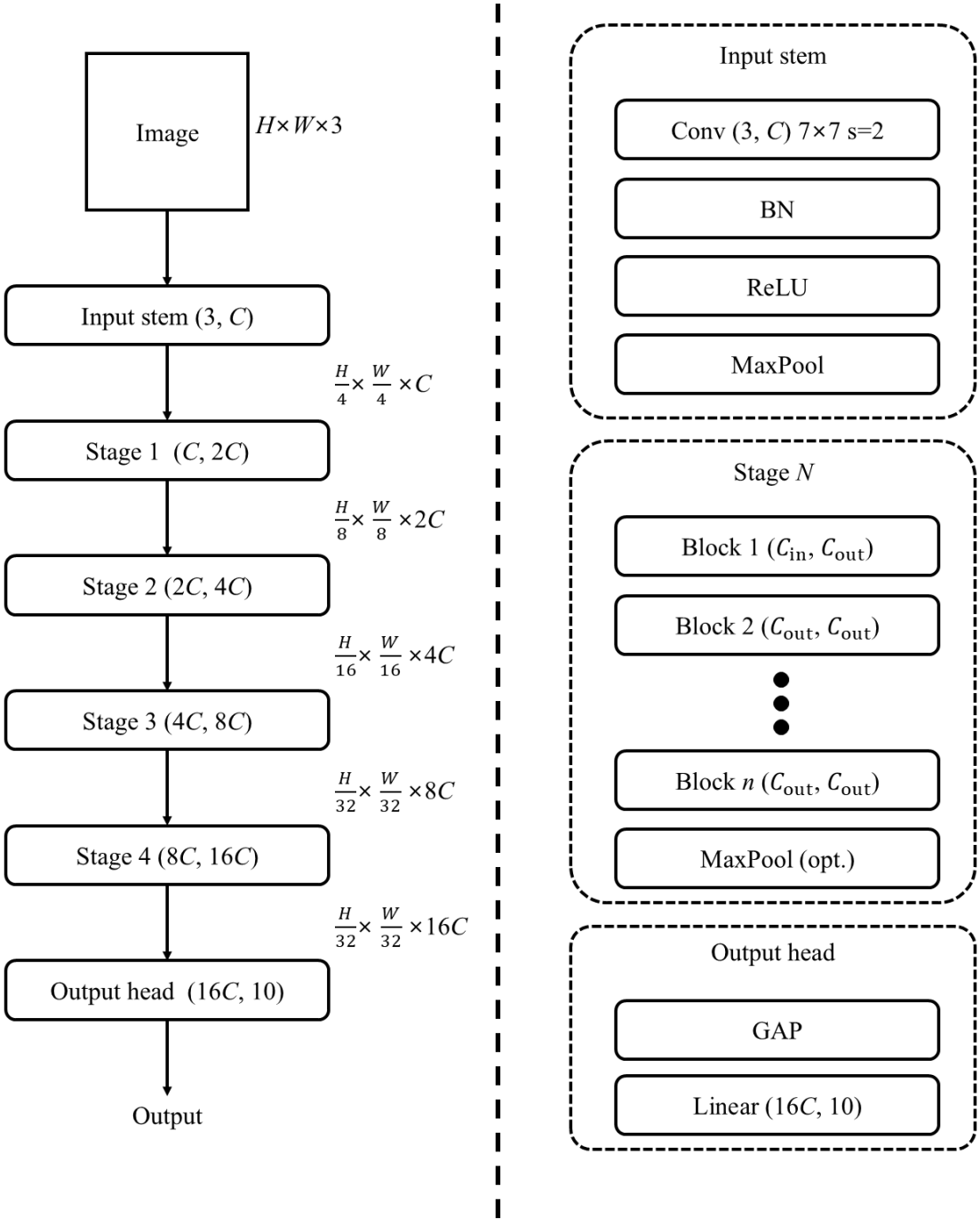

Figure 1: Overview of the network architecture. **Left:** The hierarchical structure consisting of four stages. **Right:** Module details, including the 7×7 convolution in the input stem, block-based stages, and the Global Average Pooling (GAP) layer in the output head.

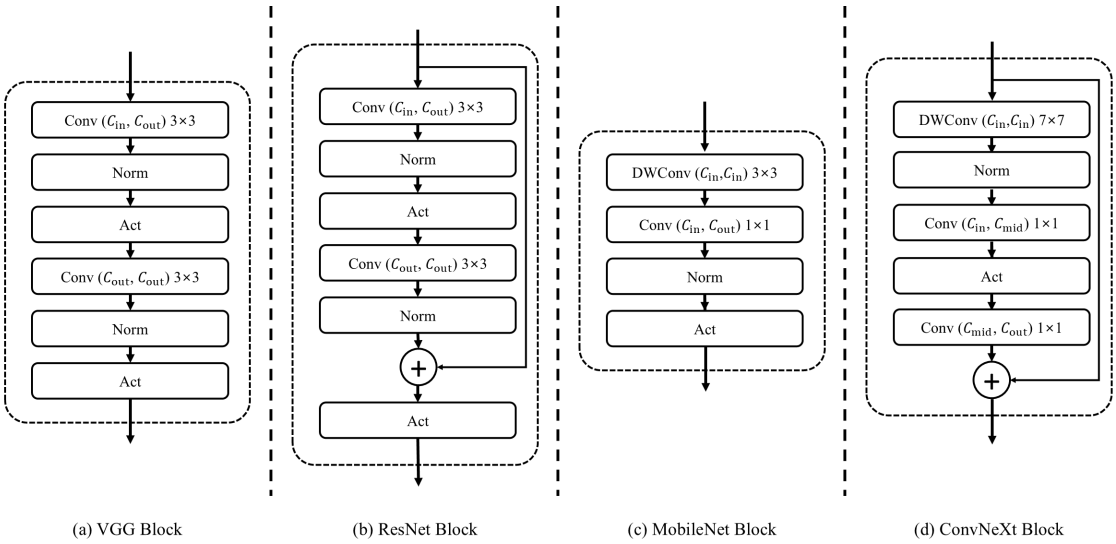

Figure 2: Detailed structures of the blocks. (a) VGG Block [16]: Comprises two 3×3 convolutional layers, each followed by normalization (Norm) and activation (Act) layers. (b) ResNet Block [7]: Extends the VGG Block by introducing skip connections, which facilitate gradient flow and improve training stability. (c) MobileNet Block [9]: Features depthwise separable convolutions, consisting of a depthwise convolution (DWConv) and a 1×1 pointwise convolution, optimized for lightweight computation. (d) ConvNeXt Block [12]: Combines depthwise and pointwise convolutions with skip connections, integrating concepts inspired by Vision Transformers.

3.3 ResNet

ResNet [7] introduces the concept of residual connections to address the vanishing gradient problem in deep networks. By allowing the gradient to flow directly through identity mappings, ResNet facilitates the training of much deeper networks. This design significantly improves performance while maintaining manageable complexity. The detailed structure of the ResNet Block is illustrated in Figure 2(b).

3.4 MobileNet

MobileNet [9] is a lightweight architecture designed for mobile and embedded devices. It utilizes depthwise separable convolutions to reduce the number of parameters and computational costs. This efficiency makes it ideal for applications requiring low-latency and low-power consumption. The detailed structure of the MobileNet Block is illustrated in Figure 2(c).

3.5 ConvNeXt

ConvNeXt [12] is a modern convolutional architecture that integrates ideas from vision transformers [5] while retaining the efficiency of convolutional blocks. By simplifying the design of convolutional layers, ConvNeXt achieves state-of-the-art performance on various benchmarks, making it a competitive choice for high-accuracy applications. The detailed structure of the ConvNeXt Block is illustrated in Figure 2(d).

4 Experiments

In this section, we describe the experimental setup and methodology used to evaluate the performance of neural network block architectures. As this study primarily focuses on inference speed, the experiments were conducted without the use of any specific dataset, such as ImageNet [15]. Instead, dummy images with three different resolutions, 64×64, 256×256, and 1024×1024, were used as input to analyze the impact of input size on inference speed across various network architectures. The channel

size (C) was consistently set to 64, as depicted in Figure 1. Consequently, the accuracy of the networks on classification tasks or datasets was not included in these experiments.

4.1 Setup

All experiments were conducted on a system equipped with an NVIDIA GeForce RTX 3090 ti GPU, running on Windows 11. The software environment included CUDA 11.8, cuDNN 8.9.7, Python 3.11, PyTorch 2.5.1, and TensorRT 8.5.3.1. Models were implemented in PyTorch, exported to ONNX format, and optimized using TensorRT for inference in a C++ environment.

To evaluate inference speed, experiments were conducted using both FP32 and FP16 precision, allowing for a comparison of computational performance across different configurations. Each configuration included variations in normalization layers—Batch Normalization (BN) [10] and Layer Normalization (LN) [2]—and activation functions—ReLU [13] and GELU [8]. These combinations were systematically tested to analyze their impact on inference speed and computational efficiency across architectures.

Each configuration was measured three times, and the average inference speed was calculated to ensure consistent and reliable results.

4.2 Comparison to Different Configurations

The inference speed of the neural network architectures was evaluated under various combinations of normalization layers (BN [10] and LN [2]) and activation functions (ReLU [13] and GELU [8]). Additionally, both FP32 and FP16 precision were tested to compare their impact on computational performance. Experiments were conducted with three different input resolutions: 64×64, 256×256, and 1024×1024, to analyze the effect of input size on inference speed. To ensure consistency across experiments, the batch size was fixed to 1. This setup enabled a fair and comprehensive comparison of inference speed across different architectures and configurations. The results for 64, 256, and 1024 resolutions are summarized in Table 1, Table 2, and Table 3, respectively. Additionally, Figure 3, Figure 4, and Figure 5 provide a detailed visualization of the inference speeds across different architectures, normalization layers, and activation functions.

4.3 Impact of Normalization Layers and Activation Functions on Inference Speed

This subsection examines the effect of normalization layers—Batch Normalization (BN) and Layer Normalization (LN)—and activation functions—ReLU and GELU—on the inference speed of various neural network block architectures. The results indicate that BN consistently outperforms LN in terms of inference speed across all tested configurations. Similarly, ReLU achieves faster inference speeds compared to GELU. These differences are especially pronounced when using FP16 precision, highlighting the critical role of optimization mechanisms in TensorRT.

For example, in MobileNet with FP16 precision, configurations using BN achieve approximately **3x, 2x, and 1.6x faster inference speeds** than those using LN at resolutions of 64×64, 256×256, and 1024×1024, respectively. A similar trend is observed when comparing ReLU to GELU, where ReLU consistently results in faster inference speeds across all architectures and resolutions.

The observed performance disparities can be attributed to TensorRT's ability to optimize operations by fusing them into a single kernel. BN and ReLU are highly compatible with TensorRT's Convolution-BatchNorm-ReLU (CBR) fusion mechanism, where these three layers are combined into a single, highly optimized operation during inference. Although ReLU is a nonlinear activation function, its computational simplicity—defined by its element-wise max operation—makes it well-suited for such fusion. In contrast, LN and GELU involve more complex element-wise computations and nonlinearities that make them less amenable to fusion. This limitation increases computational overhead and slows down inference for configurations using LN or GELU.

These findings underscore the importance of considering the compatibility of normalization layers and activation functions with optimization frameworks like TensorRT when designing neural network architectures for high-performance inference, particularly in latency-sensitive or resource-constrained environments.

Anomalous behavior was observed when Layer Normalization (LN) was used with low-resolution inputs, particularly in MobileNet. At 64×64 resolution, the inference speed with FP16 precision was slower than that with FP32 precision. This unexpected result likely arises from LN's element-wise operations, which introduce memory access overhead and limit parallelization in low-resolution scenarios. Additionally, FP16 precision may require additional computations for numerical stability, offsetting its expected speed advantages.

4.4 Impact of Block Structures on Inference Speed

This subsection examines how the structural differences in neural network blocks—VGG, ResNet, MobileNet, and ConvNeXt—affect inference speed across various configurations and input resolutions. Experimental results show that MobileNet consistently achieved the fastest inference speeds across all configurations and resolutions. Its block design, which employs depthwise separable convolutions, drastically reduces the number of multiply-accumulate operations (MACs) compared to other architectures, making it highly efficient in resource-constrained environments.

In contrast, ConvNeXt exhibited the slowest inference speeds, even with fewer parameters and MACs compared to VGG and ResNet. For example, at a resolution of 256×256, ConvNeXt was approximately **2.4 times slower** than both VGG and ResNet under FP16 precision. This disparity may arise from ConvNeXt's reliance on computationally intensive operations inspired by vision transformers, which may not be fully optimized for inference frameworks like TensorRT.

VGG and ResNet demonstrated intermediate performance. However, the relatively high parameter count and computational demand of both architectures resulted in slower inference speeds compared to MobileNet.

These findings underscore the critical role of block structure in determining inference speed. Designing neural networks for latency-sensitive applications requires balancing block structure complexity with inference framework efficiency to achieve optimal performance.

5 Conclusion

In this paper, we conducted a comprehensive evaluation of the inference speed of various neural network block architectures, including VGG, ResNet, MobileNet, and ConvNeXt, under different configurations of normalization layers, activation functions, and input resolutions. The findings provide valuable insights into the trade-offs between computational performance and architectural design, which are critical for optimizing neural networks in resource-constrained and latency-sensitive environments.

The results highlight several key observations. Batch Normalization (BN) consistently outperformed Layer Normalization (LN) in inference speed, largely due to TensorRT's ability to fuse BN with Convolution and ReLU layers into a single optimized operation. ReLU also achieved faster inference speeds than GELU, further emphasizing the importance of compatibility with optimization frameworks like TensorRT. Among the architectures, MobileNet exhibited the fastest inference speeds, benefiting from its lightweight structure and reduced multiply-accumulate operations (MACs). Conversely, ConvNeXt showed slower inference speeds despite having fewer parameters and MACs compared to VGG and ResNet, likely due to its reliance on computationally intensive operations inspired by vision transformers.

An anomalous behavior was observed with LN and FP16 precision at low resolutions, particularly in MobileNet. At 64×64 resolution, FP16 precision was slower than FP32, likely due to memory access overhead and numerical instability in LN calculations.

This study has certain limitations. The evaluation was conducted using dummy images rather than real-world datasets, which may affect the applicability of the findings. Additionally, experiments were performed on a single hardware configuration—an NVIDIA GeForce RTX 3090 ti GPU—limiting the generalizability of the results to other platforms. Finally, the analysis was confined to inference speed, excluding metrics like memory usage, energy efficiency, and accuracy in practical applications.

Future work could expand the scope by exploring a wider range of block architectures and evaluating their performance on real-world datasets. Such efforts would provide deeper insights into architectural trade-offs and enhance the applicability of these findings in practical scenarios.

Architecture	Resolution	Norm	Act	#Params	MACs	FP32	FP16
VGG	64×64	BN	ReLU	94.05 M	1.14 G	2.512 ms	0.881 ms
ResNet	64×64	BN	ReLU	94.75 M	1.15 G	2.547 ms	0.904 ms
MobileNet	64×64	BN	ReLU	10.62 M	0.14 G	1.123 ms	0.723 ms
ConvNeXt	64×64	BN	ReLU	83.7 M	1.04 G	2.179 ms	2.125 ms
VGG	64×64	LN	ReLU	94.05 M	1.14 G	3.319 ms	2.375 ms
ResNet	64×64	LN	ReLU	94.75 M	1.15 G	3.355 ms	2.357 ms
MobileNet	64×64	LN	ReLU	10.62 M	0.14 G	2.178 ms	2.29 ms
ConvNeXt	64×64	LN	ReLU	83.7 M	1.04 G	3.39 ms	3.323 ms
VGG	64×64	BN	GELU	94.05 M	1.14 G	2.583 ms	0.902 ms
ResNet	64×64	BN	GELU	94.75 M	1.15 G	2.66 ms	0.95 ms
MobileNet	64×64	BN	GELU	10.62 M	0.14 G	1.136 ms	0.756 ms
ConvNeXt	64×64	BN	GELU	83.7 M	1.04 G	2.322 ms	2.136 ms
VGG	64×64	LN	GELU	94.05 M	1.14 G	3.208 ms	2.326 ms
ResNet	64×64	LN	GELU	94.75 M	1.15 G	3.509 ms	2.305 ms
MobileNet	64×64	LN	GELU	10.62 M	0.14 G	2.231 ms	2.266 ms
ConvNeXt	64×64	LN	GELU	83.7 M	1.04 G	3.443 ms	3.398 ms

Table 1: Inference Speed and Computational Cost of Neural Network Architectures at 64×64 Resolution Under Different Configurations.

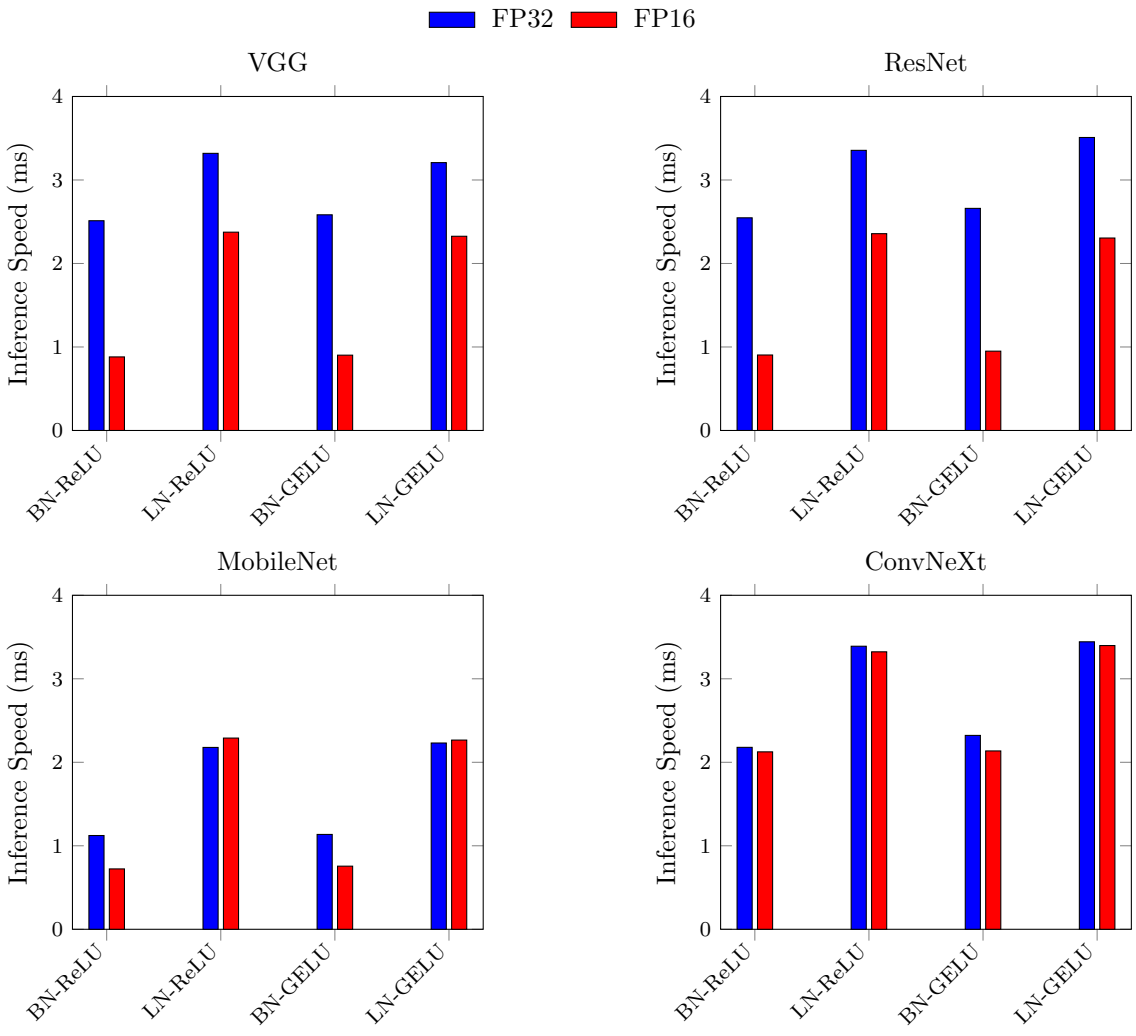

Figure 3: Inference Speed for 64×64 Resolution across Different Architectures, Normalization Layers, and Activation Functions.

Architecture	Resolution	Norm	Act	#Params	MACs	FP32	FP16
VGG	256 × 256	BN	ReLU	94.05 M	18.31 G	2.844 ms	1.091 ms
ResNet	256 × 256	BN	ReLU	94.75 M	18.45 G	2.849 ms	1.076 ms
MobileNet	256 × 256	BN	ReLU	10.62 M	2.27 G	1.155 ms	0.974 ms
ConvNeXt	256 × 256	BN	ReLU	83.7 M	16.61 G	3.361 ms	2.613 ms
VGG	256 × 256	LN	ReLU	94.05 M	18.3 G	3.74 ms	2.777 ms
ResNet	256 × 256	LN	ReLU	94.75 M	18.43 G	3.748 ms	2.689 ms
MobileNet	256 × 256	LN	ReLU	10.62 M	2.26 G	2.561 ms	2.621 ms
ConvNeXt	256 × 256	LN	ReLU	83.7 M	16.6 G	4.803 ms	4.108 ms
VGG	256 × 256	BN	GELU	94.05 M	18.31 G	3.144 ms	1.168 ms
ResNet	256 × 256	BN	GELU	94.75 M	18.45 G	3.02 ms	1.214 ms
MobileNet	256 × 256	BN	GELU	10.62 M	2.27 G	1.386 ms	1.009 ms
ConvNeXt	256 × 256	BN	GELU	83.7 M	16.61 G	3.357 ms	2.698 ms
VGG	256 × 256	LN	GELU	94.05 M	18.3 G	3.744 ms	2.667 ms
ResNet	256 × 256	LN	GELU	94.75 M	18.43 G	3.774 ms	2.756 ms
MobileNet	256 × 256	LN	GELU	10.62 M	2.26 G	2.656 ms	2.497 ms
ConvNeXt	256 × 256	LN	GELU	83.7 M	16.6 G	4.809 ms	4.305 ms

Table 2: Inference Speed and Computational Cost of Neural Network Architectures at 256 × 256 Resolution Under Different Configurations.

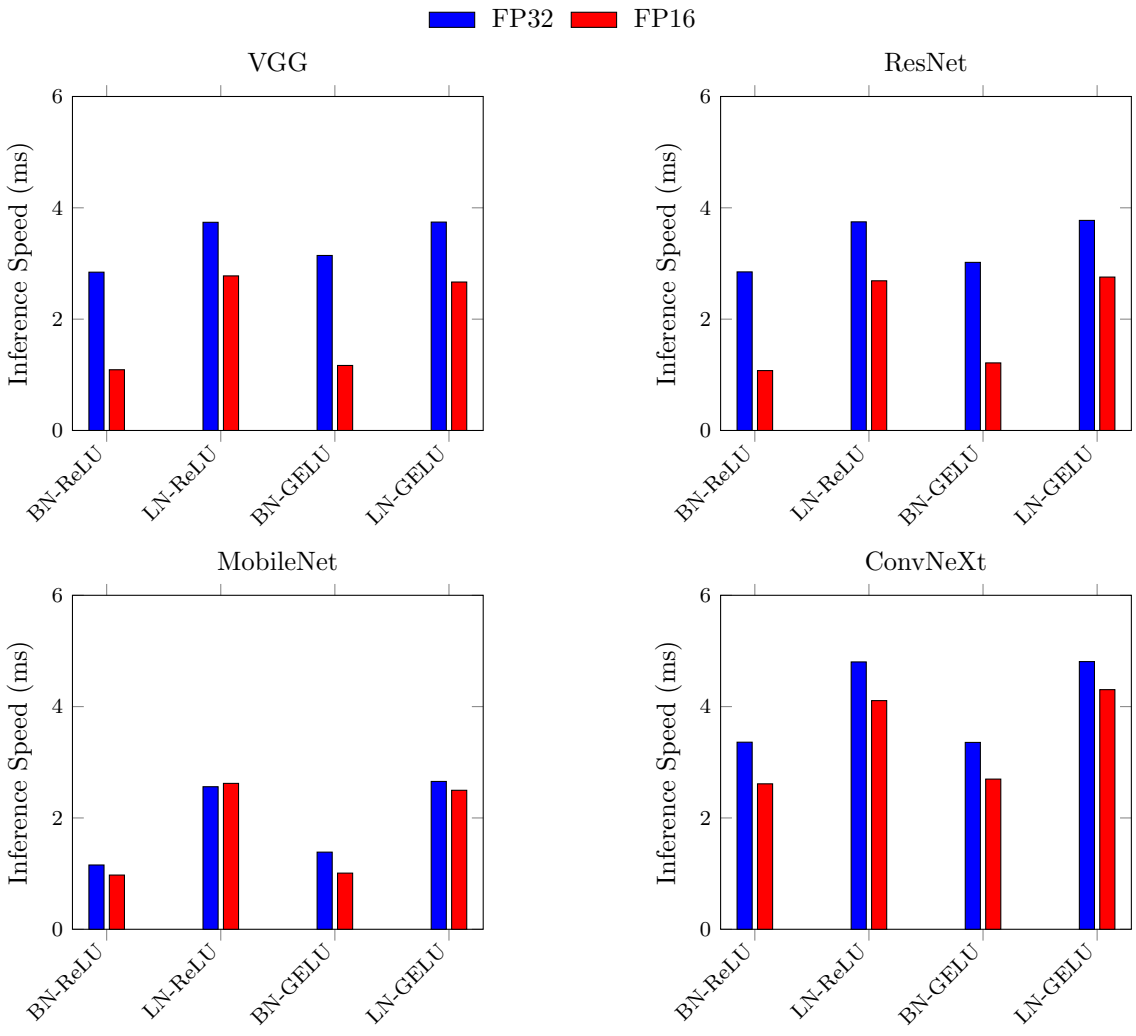

Figure 4: Inference Speed for 256×256 Resolution across Different Architectures, Normalization Layers, and Activation Functions.

Architecture	Resolution	Norm	Act	#Params	MACs	FP32	FP16
VGG	1024 × 1024	BN	ReLU	94.05 M	293.01 G	21.444 ms	10.198 ms
ResNet	1024 × 1024	BN	ReLU	94.75 M	295.16 G	21.977 ms	10.171 ms
MobileNet	1024 × 1024	BN	ReLU	10.62 M	36.38 G	9.509 ms	6.902 ms
ConvNeXt	1024 × 1024	BN	ReLU	83.7 M	265.82 G	35.197 ms	18.932 ms
VGG	1024 × 1024	LN	ReLU	94.05 M	292.76 G	27.934 ms	16.119 ms
ResNet	1024 × 1024	LN	ReLU	94.75 M	294.91 G	27.864 ms	15.715 ms
MobileNet	1024 × 1024	LN	ReLU	10.62 M	36.12 G	15.186 ms	11.708 ms
ConvNeXt	1024 × 1024	LN	ReLU	83.7 M	265.59 G	43.007 ms	22.067 ms
VGG	1024 × 1024	BN	GELU	94.05 M	293.01 G	23.614 ms	11.033 ms
ResNet	1024 × 1024	BN	GELU	94.75 M	295.16 G	23.597 ms	11.236 ms
MobileNet	1024 × 1024	BN	GELU	10.62 M	36.38 G	11.069 ms	7.684 ms
ConvNeXt	1024 × 1024	BN	GELU	83.7 M	265.82 G	35.498 ms	19.485 ms
VGG	1024 × 1024	LN	GELU	94.05 M	292.76 G	28.244 ms	16.089 ms
ResNet	1024 × 1024	LN	GELU	94.75 M	294.91 G	28.465 ms	15.954 ms
MobileNet	1024 × 1024	LN	GELU	10.62 M	36.12 G	15.215 ms	12.13 ms
ConvNeXt	1024 × 1024	LN	GELU	83.7 M	265.59 G	43.04 ms	24.466 ms

Table 3: Inference Speed and Computational Cost of Neural Network Architectures at 1024 × 1024 Resolution Under Different Configurations.

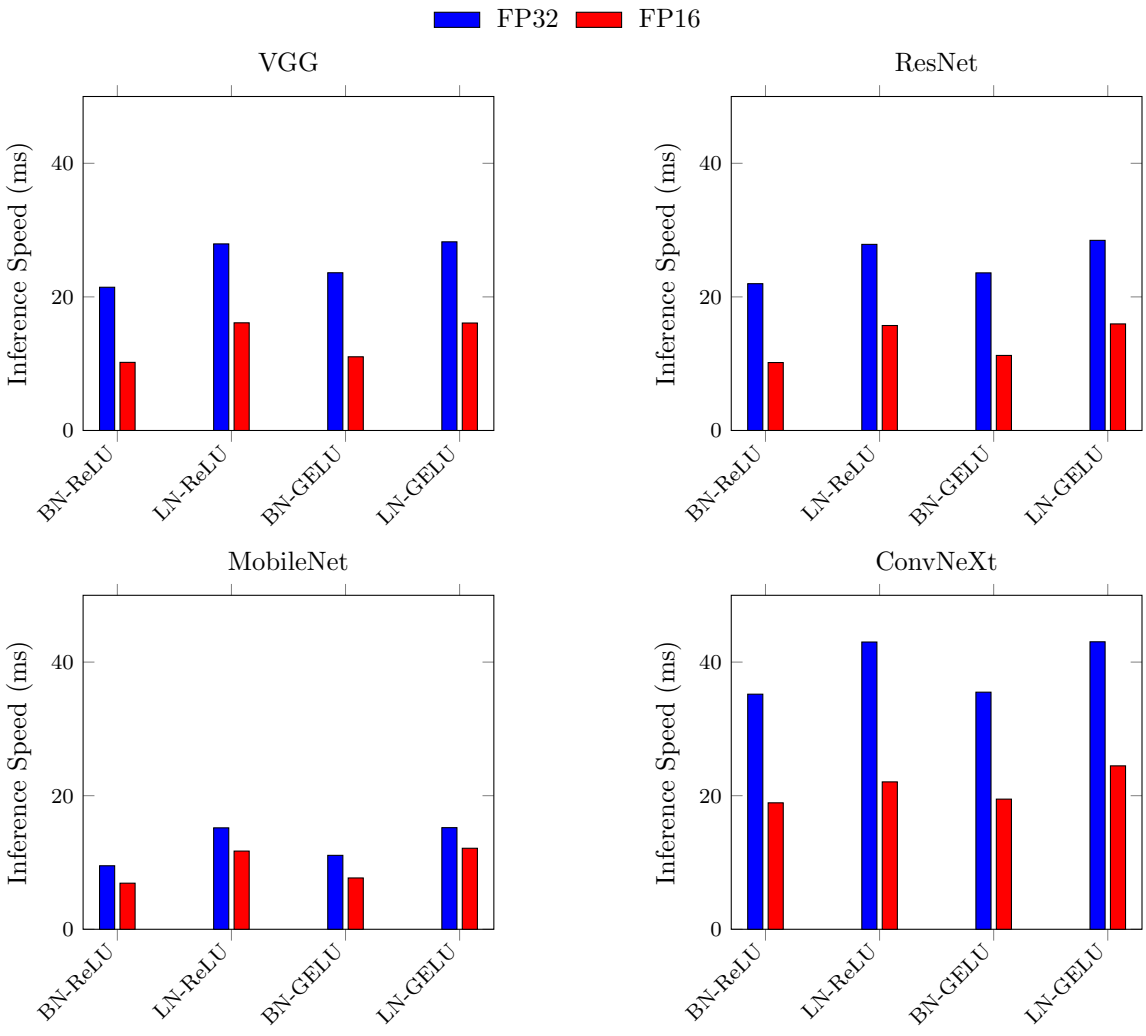

Figure 5: Inference Speed for 1024 × 1024 Resolution across Different Architectures, Normalization Layers, and Activation Functions.

References

[1] Shahin Atakishiyev, Mohammad Salameh, Hengshuai Yao, and Randy Goebel. Explainable artificial intelligence for autonomous driving: A comprehensive overview and field guide for future research directions. *IEEE Access*, 2024.

[2] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[3] Mariusz Bojarski, Davide Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *arXiv preprint arXiv:1604.07316*, 2016.

[4] Thao Do, Yalew Tolcha, Tae Joon Jun, and Daeyoung Kim. Smart inference for multidigit convolutional neural network based barcode decoding. In *Proceedings of the 25th International Conference on Pattern Recognition (ICPR)*, pages 3019–3026. IEEE, 2021.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[6] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3396. IEEE, 2017.

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

[8] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[9] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[10] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32th International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[11] Shashi Bhushan Jha and Radu F. Babiceanu. Deep cnn-based visual defect detection: Survey of current literature. *Computers in Industry*, 148:103911, 2023.

[12] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11976–11986, 2022.

[13] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, pages 807–814, 2010.

[14] Ricardo Silva Peres, Xiaodong Jia, Jay Lee, Keyi Sun, Armando Walter Colombo, and Jose Barata. Industrial artificial intelligence in industry 4.0: Systematic review, challenges, and outlook. *IEEE Access*, 8:220121–220139, 2020.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115:211–252, 2015.

[16] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[17] Yuxiao Zhou and Kecheng Yang. Exploring tensorrt to improve real-time inference for deep learning. In *2022 IEEE 24th International Conference on High Performance Computing and Communications; 8th International Conference on Data Science and Systems; 20th International Conference on Smart City; 8th International Conference on Dependability in Sensor, Cloud and Big Data Systems and Application (HPCC/DSS/SmartCity/DependSys)*, pages 2011–2018, 2022.