UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

Visión Electrónica
https://doi.org/10.14483/issn.2248-4728

VISIÓN ELECTRÓNICA

A RESEARCH VISION

GitHub Extension for FlexiPrune: Automating Pruning Distribution in CNNs

Extensión en GitHub para FlexiPrune: automatización de la distribucón de poda en CNNs

Dora Ballesteros[1] , Laura Catalina Rozo[2] , Cesar G. Pachòn[3]

INFORMACIÓN DEL ARTÍCULO

Historia del artículo:
Enviado: 05/09/2024
Recibido: 09/09/2024
Aceptado: 07/11/2024

Keywords:
Convolutional Neutral Network (CNN)
FlexiPrune
Global Pruning Gate (GPR)
Pruning
Pruning Distribution (PD)
Pruning Rate (PR)

Palabras clave:
FlexiPrune
Distribuciòn de poda
Poda
Redes neuronales convolucionales
Tasa de poda
Tasa de poda global

ABSTRACT

FlexiPrune is an open-source library developed in Python using PyTorch, available on GitHub. It supports the compression of convolutional neural network (CNN) models for image classification by applying various pruning strategies. The library includes three core modules: model training, pruning, and evaluation. In the pruning module, users can choose from multiple methods and pruning distribution types, such as homogeneous, bottom-up, and top-down. However, a major limitation is the need to manually specify the pruning rate (PR) for each layer, which is time-consuming and error-prone, especially for non-uniform distributions. To address this issue, we introduce a fully compatible Python module that automates the calculation of per-layer PRs based on a global pruning rate (GPR) and a selected distribution type (PD), from PD1 (homogeneous) to PD5 (top-up/top-down). The module also estimates the resulting reduction in floating-point operations (FLOPs) and parameters, ensuring consistent computational effort across distributions. We evaluated the module on five CNN architectures (AlexNet, VGG11, VGG13, VGG16, VGG19) and six global pruning rate values (GPRs) (10 % to 60 %). Results show that PD2 consistently leads to the highest parameter reduction, while PD3 gives the lowest. FLOP reductions remain nearly identical across distributions for the same GPR, with standard deviations below 1. These outcomes confirm the module's robustness and its contribution to automating and fairly comparing pruning strategies in FlexiPrune.

RESUMEN

FlexiPrune es una librería de código abierto desarrollada en Python con PyTorch, disponible en GitHub. Permite comprimir modelos de redes neuronales convolucionales (CNN) para clasificación de imágenes mediante distintas estrategias

1. Bsc. Electronic Engineering, Universidad Industrial de Santander, Colombia. PhD. Electronic Engineering, Universidad Politécnica de Cataluña. Current position, Titular Professor, Universidad Militar Nueva Granada, Colombia. E-mail: dora.ballesteros@unimilitar.edu.co ORCID: https://orcid.org/0000-0003-3864-818X.
2. Bsc(c). Telecommunications Engineering, Universidad Militar Nueva Granada, Colombia. E-mail: est.laura.rozo1@unimilitar.edu.co.
3. Bsc. Mechatronics Engineering, Universidad Piloto, Colombia. MSc. Mechatronics Engineering, Universidad Militar Nueva Granada, Colombia. PhD. (c) at Engineering, Universidad Militar Nueva Granada, Colombia. E mail: est.cesar.pachon@unimilitar.edu.co ORCID: https://orcid.org/0000-0001-8899-8298

Cite this article as: D. Ballesteros, L. C. Rozo, C. G. Pachón, "GitHub Extension for FlexiPrune: Automating Pruning Distribution in CNNs" Visión Electrónica, vol. 19, no. 1, 2025.

de poda, e incluye tres módulos principales: entrenamiento, poda y evaluación. En el módulo de poda, los usuarios pueden elegir entre varios métodos y tipos de distribución, como homogénea, descendente o ascendente. No obstante, una limitación importante es que el usuario debe ingresar manualmente la tasa de poda (PR) para cada capa, lo cual es tedioso y propenso a errores, especialmente en distribuciones no uniformes. Para resolver esta limitación, se propone un módulo adicional totalmente compatible con FlexiPrune, que calcula automáticamente las PR por capa a partir de una tasa de poda global (GPR) y un tipo de distribución (PD), desde PD1 (homogénea) hasta PD5 (ascendente/descendente). Además, estima la reducción en operaciones de punto flotante (FLOPs) y en parámetros, garantizando una carga computacional equivalente entre distribuciones. El módulo fue evaluado en cinco arquitecturas (AlexNet, VGG11, VGG13, VGG16 y VGG19) y seis valores de GPR (10 % al 60 %). Los resultados muestran que la distribución PD2 logra la mayor reducción de parámetros, mientras que PD3 obtiene la menor. Las reducciones de FLOPs se mantienen prácticamente iguales para todas las distribuciones bajo el mismo GPR, con desviaciones estándar por debajo de 1. Estos resultados validan el diseño del módulo y fortalecen su utilidad para automatizar y comparar estrategias de poda en FlexiPrune.

1. Introduction

At the end of 2024, FlexiPrune was released as an open-source library written in Python and based on PyTorch [1, 2], designed for compressing image classification models using pruning techniques in convolutional neural networks (CNNs). This modular tool includes three main components: training, pruning, and model evaluation. FlexiPrune allows users to select different pruning methods—such as Random [3–6], Weight [7, 10], and SeNPIS-Faster [11]—as well as to define global pruning rates (GPRs) and pruning Distribution types (PDs), enabling the application of suitable compression policies for specific models and datasets. This flexibility facilitates comparisons between multiple pruning policy combinations, which is particularly valuable given that no single configuration is optimal across all scenarios. Previous studies have shown that, even within the same classification problem, significant performance differences can arise in compressed models when varying the GPR value [12–16].

However, FlexiPrune presents a key limitation: users must manually input the pruning rate (PR) for each layer of the network, which becomes especially complex in cases involving non homogeneous distributions. This process is not only time-consuming but also error-prone, and may hinder fair comparisons between models if equivalent FLOPs reductions are not achieved. A lack of familiarity with the specific calculations required for different layer types (convolutional, pooling, fully connected) can significantly reduce the tool's accessibility and usability.

To overcome this limitation, we propose a new complementary module for FlexiPrune that automates the generation of layer-wise PR lists, based on a user-defined GPR value and selected pruning distribution type (PD). The module is compatible with sequential architectures such as AlexNet, VGG11, VGG13, VGG16, and VGG19, and supports five distribution types: homogeneous (PD1), descending (PD2), ascending (PD3), descending/ascending (PD4), and ascending/descending (PD5). In this scheme, fully connected layers are pruned according to the specified GPR, and the final classification layer remains unchanged.

In addition to automatically calculating PRs for each layer, the module estimates the total number of parameters and FLOPs in the pruned model, as well as the reductions compared to the original model. This feature allows users to evaluate multiple pruning policies under comparable conditions and make informed decisions about trade-offs between compression and performance. The integration of this module significantly enhances FlexiPrune's capabilities by eliminating the need for manual calculations and facilitating broader and more rigorous experimental applications.

2. Materials and Methods

The following describes the current state of the FlexiPrune library and how the proposed module would be integrated into it.

2.1. FlexiPrune Library

FlexiPrune is a PyTorch-based library that includes the following modules: custom dataset, model training (train_model), model parameters (ModelParams), model pruning (prune_model), and model evaluation (evaluate_models). See Figure 1.

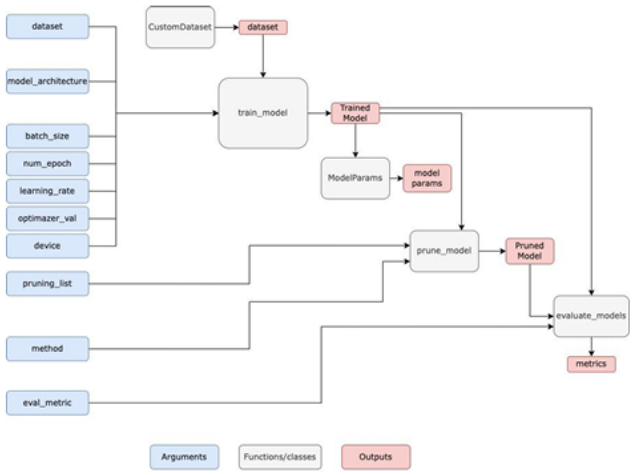


Figure 1: General architecture of the FlexiPrune library. Source: Own.

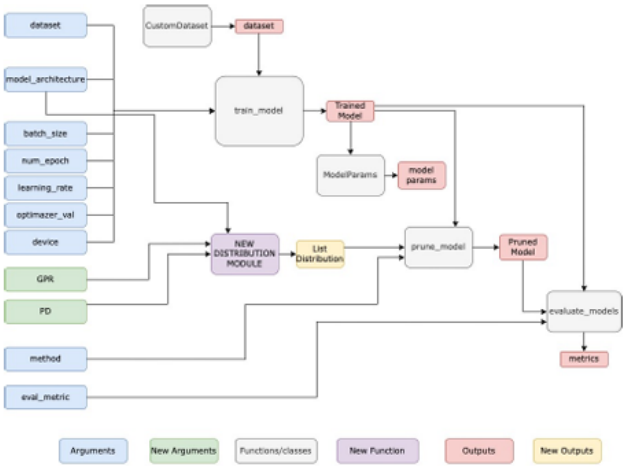


Figure 2: General architecture of the FlexiPrune library with the proposed new module. Source: Own.

Users can choose a dataset (e.g., CIFAR10) and a network (e.g., VGG16) and then train the model for a specified number of epochs. After training, the model is pruned using one of the methods provided in the library (i.e., Random, Weight, SeNPIS-Faster), or a custom method (see [17–20]) and a selected pruning distribution. The pruned model is then evaluated against the original model in terms of accuracy (for balanced datasets) or F1-score (for unbalanced datasets), as well as in FLOPs and parameters. When multiple methods and distribution types are selected, the library can generate performance plots for the metrics.

In the pruning module, users must select the pruning method and input the pruning rate (PR) for each convolutional and fully connected (FC) layer in the network (i.e., the pruning list). For example, for VGG16, the following pruning list could be entered: 20, 40, 46, 46, 50, 50, 50, 60, 60, 60, 68, 70, 70, 50, and 50. These values result in a global pruning rate (GPR) of 50 % with an ascending pruning distribution. However, users needed expertise in estimating FLOPs and CNN parameters to determine each PR value based on a desired GPR and distribution type. Otherwise, comparisons with other pruned models using the same GPR and method, but a different distribution, would not be fair. Thus, manually entering PRs became a potential limitation of the FlexiPrune library.

2.2. FlexiPrune Library with the Proposed Module

Based on the above, a new module compatible with FlexiPrune has been proposed (see Figure 2 – New distribution module).

This module computes the The pruning rate values (PRs) for each convolutional and fully connected (FC) layer of the network (i.e., the distribution list), using the GPR and the selected distribution type (i.e., PD1 = homogeneous, PD2 = descending, PD3 = ascending, PD4 = descending/ascending, and PD5 = ascending/descending) for a specific network (model_architecture). Additionally, it estimates the expected reduction in FLOPs and parameters. With the proposed module, FlexiPrune users would no longer need to manually enter The pruning rate values (PRs); instead, these would be automatically calculated within the pruning library.

2.3. Task of the proposed module

The proposed module performs the following tasks: (1) PR calculation per layer, (2) calculation of retained filters after pruning, (3) FLOPs estimation, and (4) parameter estimation. Each of these tasks is described below.

2.3.1. The Pruning Rate values (PRs)

This step computes the The pruning rate values (PRs) for each of the convolutional and FC layers. For example, in the case of VGG16, 13 PRs are required for the convolutional layers and 2 for the FC layers. First, the 15 The pruning rate values (PRs) are assigned when GPR = 50 %, for the different PDs (see Table 1).

It is important to note that regardless of the selected PD type, the number of pruned neurons in the FC layers corresponds to the GPR value (in VGG16, this applies to the last two layers). The values presented in Table 1 are not the only ones that satisfy the GPR and selected distribution type, but

they are the ones used by the new module to standardize the number of FLOPs across the pruned models.

Table 1: Pruning distributions and PR per layer

Pruning Distribution	PR per layer
$PD1_{50}$	{50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50, 50}
$PD2_{50}$	{20, 40, 46, 46, 50, 50, 50, 60, 60, 60, 68, 70, 70, 50, 50}
$PD3_{50}$	{70, 70, 62, 62, 50, 50, 50, 40, 40, 40, 34, 34, 32, 50, 50}
$PD4_{50}$	{30, 30, 48, 48, 60, 60, 60, 60, 60, 60, 30, 30, 30, 50, 50}
$PD5_{50}$	{66, 64, 50, 50, 40, 40, 40, 50, 50, 50, 64, 66, 66, 50, 50}

Source: Own

From the above values, the PRs for any GPR can be obtained by applying a weighting between the (new) GPR and the reference GPR (which in this case is GPR = 50 %), as follows: $PDi_GPR = PDi_50 * GPR/50$. For example, when GPR = 20, then GPR/50 = 20/50 = 0.4, and $PDi_20 = PDi_50 * 0{,}4$.

2.3.2. Number of Retained Filters

It calculates the number of filters retained after pruning for the specific network. It uses the list of pruning rates obtained in the previous step. Based on those values, the number of retained filters is computed using the formula: retained_filters = filters × 1 – (PDi_GPR / 100). For example, if GPR = 20 and PD = 4, then: retained_filters = filters × 1 – (PDi_20 / 100), or in other words: retained_filters = filters × 1 – [(PD4_50 × GPR / 50) / 100] = filters × 1 – (PD4_50 × 0.4 / 100). For VGG16, which has 64 filters in the first layer, the number of retained filters in that layer will be: 64 × 1 – (30 × 0.4 / 100) = 64 × 1 – 0.12 = 64 × 0.88 = 56.

Table 2: Retained filters per layer for each pruning distribution

Pruning distribution	Retained filters
Before the pruning process	{64, 64, 128, 128, 256, 256, 256, 512, 512, 512, 512, 512, 512, 4096, 4096}
$PD1_{20}$	{51, 51, 102, 102, 204, 204, 204, 409, 409, 409, 409, 409, 409, 3276, 3276}
$PD2_{20}$	{58, 53, 104, 104, 204, 204, 204, 389, 389, 389, 373, 368, 368, 3276, 3276}
$PD3_{20}$	{46, 46, 97, 97, 204, 204, 204, 430, 430, 430, 445, 445, 450, 3276, 3276}
$PD4_{20}$	{56, 56, 103, 103, 194, 194, 194, 389, 389, 389, 450, 450, 450, 3276, 3276}
$PD5_{20}$	{47, 48, 102, 102, 215, 215, 215, 409, 409, 409, 384, 378, 378, 3276, 3276}

Source: Own

2.3.3. Number of Parameters and FLOPs

Once the number of filters or neurons retained after pruning is known, the corresponding equations for calculating FLOPs and parameters are applied to the convolutional and fully connected (FC) layers. The details of these equations can be found in [12].

FLOPs for convolutional layers are computed based on the filter size and the output shape (before convolution), while FLOPs for pooling layers depend only on the output shape. On the other hand, the parameters of convolutional layers depend solely on the filter size, and pooling layers do not contribute any parameters.

For example, for the first five layers of VGG16 (i.e., from block1_conv1 to block2_conv2), the number of FLOPs and parameters before and after pruning (with GPR = 20 % and PD4) is shown in Table 3.

Table 3. FLOs and Parameters: VGG16 (partial). GPR= 20%, PD4.

	block1_conv1	block1_conv2	block1_pool	block2_conv1	block2_conv2
FLOPs (original vs pruned)	173.408.256	3.699.376.128	802.816	1.849.688.064	3.699.376.128
	151.732.224	2.832.334.848	702.464	1.302.368.256	2.395.427.328
Parameters (original vs. pruned)	1.792	36.928	0	73.856	147.584
	1.568	28.280	0	52.015	95.584

Fuente: authors

For VGG16, the total number of FLOPs and parameters in the original network is 30,933,948,928 and 134,301,514, respectively. For GPR = 20 % and PD4, these values are 19,608,314,278 and 92,668,552, respectively.

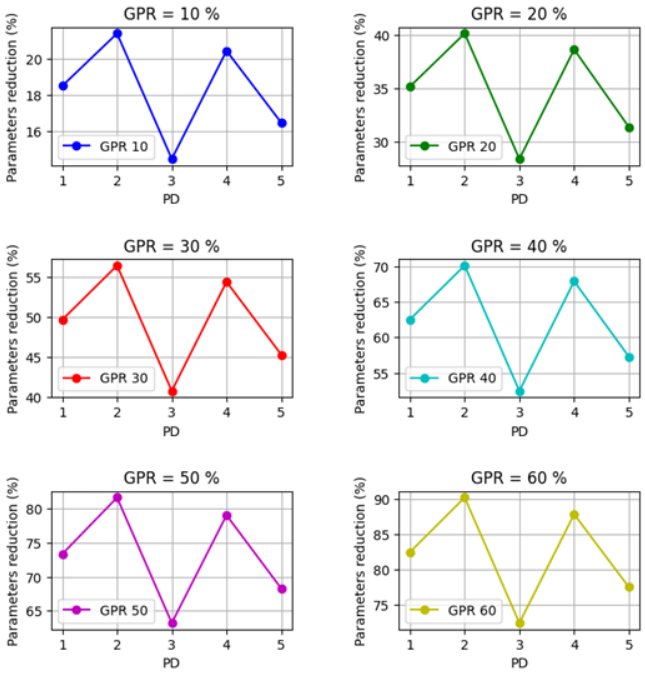
3. Results and Discussion

The results obtained for the five types of sequential networks included in the proposed module (AlexNet, VGG11, VGG13, VGG16, and VGG19), and six GThe pruning rate values (PRs) (10 %, 20 %, 30 %, 40 %, 50 %, and 60 %), with the five pruning distribution types (PD1 to PD5), are presented below.

In each case, the FLOPs reduction remains statistically the same when the network architecture and GPR are fixed, with variations only occurring in the parameter reduction values.

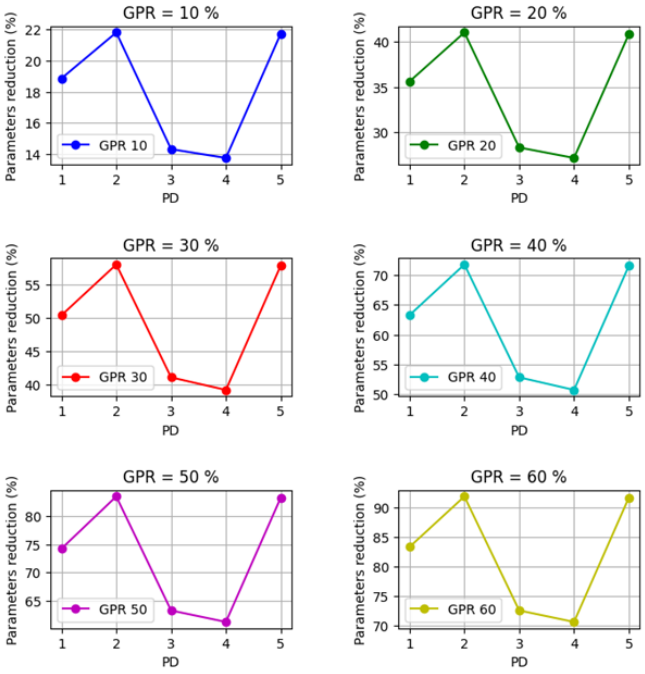
The parameter reduction results for AlexNet are shown in Figure 3. It can be observed that a clear “pattern” emerges: the network exhibits the greatest parameter reduction with PD2 and the least with PD5. Additionally, as the GPR increases, the reduction in parameters also increases.

Figure 3: Parameter Reduction (%). Network = AlexNet, GPR: 10 %,... 60 %



Source: Own (New module in FlexiPrune).

Figure 5: Parameter Reduction (%). Network = VGG11, GPR: {10 %, ..., 60 %}.



Source: Own (New module in FlexiPrune).

Now, in terms of FLOPs reduction, the results are presented in Figure 4. It can be observed that the standard deviation of the reduction is very small (close to zero), which aligns with the design criterion of obtaining pruned networks with the "same" FLOPs reduction.

Figure 4: FLOPs Reduction (%). Network = AlexNet, GPR: 10 %,... 60 %

GPR	Average (FLOPs reduction)	Max	Min	Standard Deviation
10	18.127116	18.508378	17.485668	0.397819
20	34.466913	34.996999	33.451260	0.621638
30	48.967934	49.394512	47.987932	0.558577
40	61.491116	62.028860	60.822126	0.470271
50	72.610276	72.692658	72.526399	0.061321
60	81.235441	81.873291	80.466580	0.554131

Source: Own (New module in FlexiPrune).

For the second network, VGG11, the results are presented in Figure 5 (parameter reduction) and Figure 6 (FLOPs reduction).

Figure 6: FLOPs Reduction (%). Network = VGG11, GPR: {10 %, ..., 60 %}.

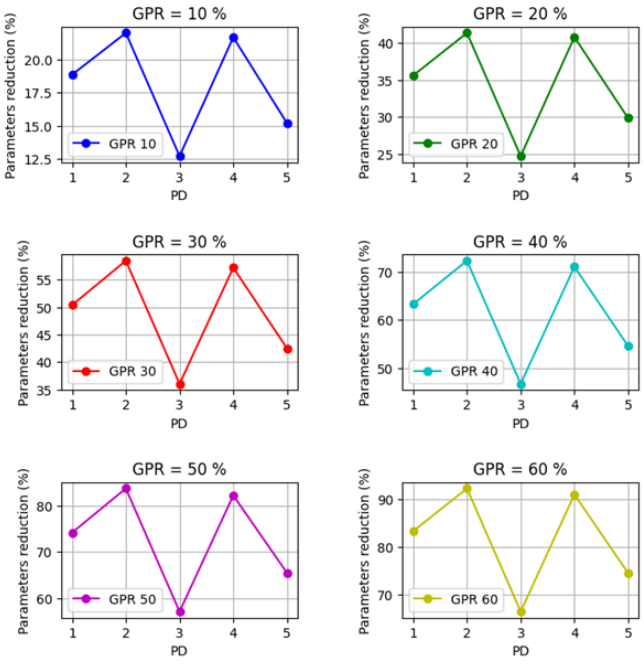
GPR	Average (FLOPs reduction)	Max	Min	Standard Deviation
10	19.168558	19.332629	18.897272	0.194713
20	36.152722	36.476939	35.683085	0.300758
30	51.025659	51.384824	50.818615	0.219584
40	63.804095	63.979048	63.512510	0.184768
50	74.690473	74.701051	74.677359	0.009781
60	83.176667	83.883762	82.783960	0.433196

Source: Own (New module in FlexiPrune).

According to Figure 5 and Figure 6, the pruning distribution type that results in the lowest parameter reduction is PD4, while PD2 achieves the highest reduction. Similar to the previous network, the standard deviation values obtained are low, in line with the design objective.

We now continue with the VGG13 network, whose results are presented in Figure 7 and Figure 8.

Figure 7: Parameter Reduction (%). Network = VGG13, GPR: {10 %, . . . , 60 %}.



Source: Own (New module in FlexiPrune).

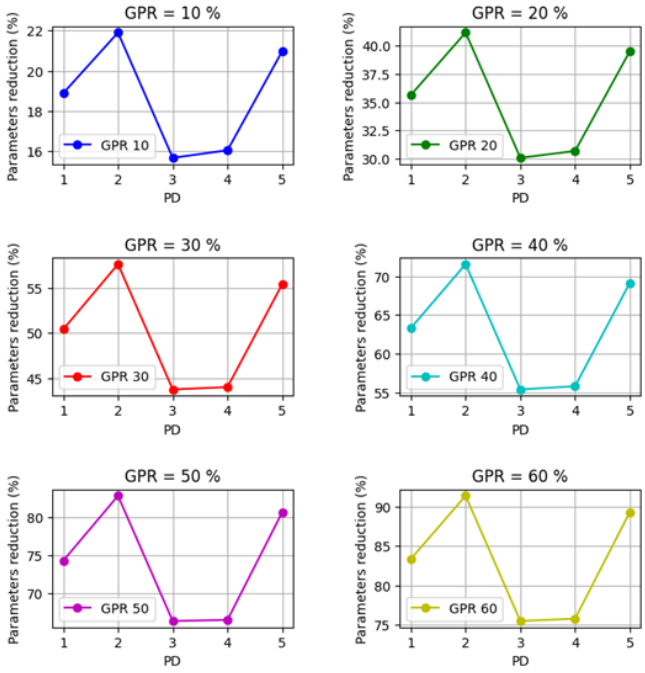
Figure 8: FLOPs Reduction (%). Network = VGG13, GPR: {10 %, . . . , 60 %}.

GPR	Average (FLOPs reduction)	Max	Min	Standard Deviation
10	19.431552	19.731590	19.170277	0.205645
20	36.813655	37.574269	36.275249	0.481395
30	51.396229	51.826107	50.797568	0.392602
40	64.266787	64.529696	63.913634	0.233784
50	74.820226	74.855910	74.798846	0.022145
60	82.846164	84.063428	82.217035	0.725212

Source: Own (New module in FlexiPrune).

In the case of VGG13, the distribution type that results in the lowest parameter reduction is PD3, while once again, PD2 yields the highest reduction. On the other hand, in terms of FLOPs reduction, the standard deviation values remain below one.We now proceed with the results for VGG16, which are presented in Figure 9 and Figure 10.

Figure 9: Parameter Reduction (%). Network = VGG16, GPR: {10 %, . . . , 60 %}.



Source: Own (New module in FlexiPrune).

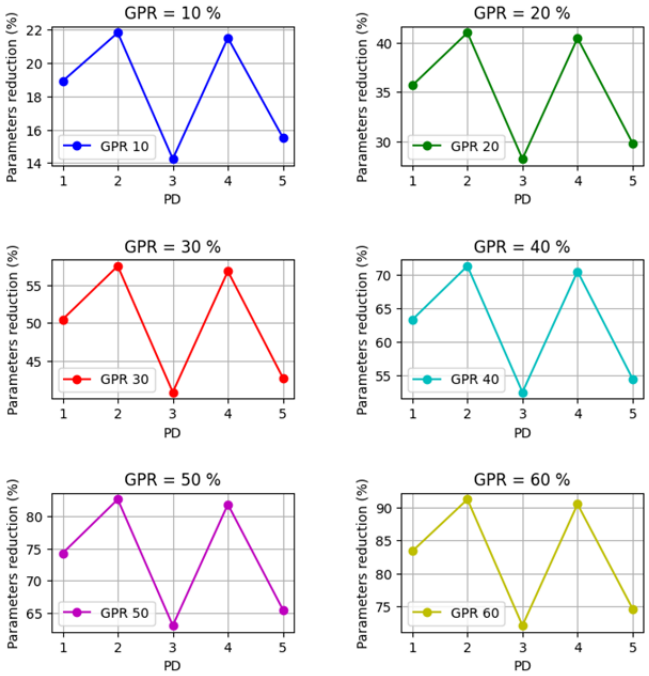
Figure 10: FLOPs Reduction (%). Network = VGG16, GPR: {10 %, . . . , 60 %}.

GPR	Average (FLOPs reduction)	Max	Min	Standard Deviation
10	19.423981	19.697346	19.325701	0.160328
20	36.456520	36.888489	36.141940	0.294624
30	51.369228	51.466059	51.276704	0.074779
40	64.178209	64.317274	64.090846	0.091387
50	74.859368	74.867144	74.849445	0.007863
60	83.458194	84.079926	83.116293	0.382540

Source: Own (New module in FlexiPrune).

In the case of VGG16, the pruning distribution that results in the lowest parameter reduction is PD3, while PD2 once again achieves the highest reduction. In terms of FLOPs reduction, the standard deviation is very low for all evaluated GThe pruning rate values (PRs), even falling below 0.5. Finally, the results for VGG19 are presented in Figure 11 and Figure 12.

Figure 11: Parameter Reduction (%). Network = VGG19, GPR: {10 %, . . . , 60 %}.



Source: Own (New module in FlexiPrune).

Figure 12: FLOPs Reduction (%). Network = VGG19, GPR: {10 %, . . . , 60 %}.

GPR	Average (FLOPs reduction)	Max	Min	Standard Deviation
10	19.035925	19.405977	18.779253	0.254328
20	36.142049	36.296715	35.874463	0.176428
30	50.981973	51.285200	50.678965	0.271847
40	63.911725	64.196399	63.718300	0.198513
50	74.878879	74.906125	74.851970	0.021968
60	83.229130	84.089429	82.844487	0.491891

Source: Own (New module in FlexiPrune).

According to Figure 11, the distribution that achieves the greatest parameter savings is PD2, while the least reduction is obtained with PD3. Regarding FLOPs reduction, the standard deviation remained below 0.5 across all evaluated GThe pruning rate values (PRs), reflecting a high degree of consistency among the distributions.

When examining the overall behavior of the five evaluated architectures (AlexNet, VGG11, VGG13, VGG16, and VGG19), it is clearly observed that the highest parameter reduction for each GPR value is consistently achieved with the PD2 distribution (i.e., descending pruning). In contrast, the lowest reduction typically occurs with PD3 (ascending pruning). This trend is consistent across all architectures, with the only exception being VGG11, where the lowest reduction is seen with PD4.

As the GPR increases, the parameter reduction also increases, and the differences between distributions become more noticeable, especially from GPR 40 % onwards. This suggests that the impact of the pruning distribution type becomes more significant under more aggressive pruning levels.

On the other hand, in terms of FLOPs reduction, the module ensures very similar results for the same GPR value and network, regardless of the chosen distribution. This consistency is reflected in the very low standard deviation values (<1 in all cases, and <0.5 for networks like VGG16 and VGG19), confirming that the module meets its objective of maintaining uniform computational load, thus enabling fair comparisons between different pruning policies.

4. Conclusions

In conclusion, the development of a module that calculates the The pruning rate values (PRs) for sequential CNN layers (e.g., VGG16) across various pruning distribution types (specifically PD1 to PD5) represents a significant advancement in pruning policy selection by incorporating not only the pruning method but also the pruning distribution type. This module, compatible with the FlexiPrune library, provides a low-computational-cost solution that enables users to easily compute The pruning rate values (PRs) based on the selected GPR, pruning distribution type, and CNN architecture. By automatically generating pruning distribution lists rather than requiring manual input, it overcomes a previous limitation of the FlexiPrune library.

In this way, FlexiPrune users can now automatically perform all tasks—from model training to pruning with various pruning policies—and compare the performance of compressed models under constant FLOPs conditions. The results obtained show that the PD2 distribution type consistently achieves the highest parameter reductions, while PD3 tends to produce the lowest. Moreover, a growing reduction pattern is preserved as the GPR increases. The low standard deviation observed in FLOPs (less than 1 in all cases) validates the proposed approach, as it enables the comparison of compressed models using homogeneous efficiency metrics, thereby strengthening informed decision-making across different network architectures.

Acknowledgments This work was supported by the Universidad Militar Nueva Granada-Vicerrectoria de Investigaciones, under project INV-ING-3947.

Referencias

[1] C.G. Pachon, J.O. Pinzon-Arenas, and D. Ballesteros, "FlexiPrune: A Pytorch tool for flexible CNN pruning policy selection," *SoftwareX*, vol. 27, p. 101858, Sep. 2024. https://doi.org/10.1016/j.softx.2024.101858

[2] C.G. Pachon, J.O. Pinzon-Arenas, and D. Ballesteros, "FlexiPrune," 2024. [Online]. Available: https://github.com/DEEP-CGPS/FlexiPrune/

[3] H. Cheng, M. Zhang, and J.Q. Shi, "A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 10558–10578, 2024. https://doi.org/10.1109/TPAMI.2024.3447085

[4] S.K. Yeom, P. Seegerer, S. Lapuschkin, and A. Binder, "Pruning by explaining: A novel criterion for deep neural network pruning," *Pattern Recognition*, vol. 115, p. 107899, 2021. https://doi.org/10.1016/j.patcog.2021.107899

[5] S. Frankle and M. Carlin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2019. https://arxiv.org/abs/1803.03635

[6] R. Li, X. Dai, Y. Chen, and D. Tao, "Random Pruning: Channel Pruning via Random Selection," *Pattern Recognition Letters*, vol. 138, pp. 377–383, 2020. https://doi.org/10.1016/j.patrec.2020.09.025

[7] S. Han, J. Pool, J. Tran, and W.J. Dally, "Learning both Weights and Connections for Efficient Neural Networks," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, 2015. https://proceedings.neurips.cc/paper_files/paper/2015/hash/ae0eb3eed39d2bcef4622b2499a05fe6-Abstract.html

[8] S. Vadera and S. Ameen, "Methods for Pruning Deep Neural Networks," *IEEE Access*, vol. 10, pp. 63280–63300, 2022. https://ieeexplore.ieee.org/document/9795013

[9] Y. He, X. Zhang, and J. Sun, "Channel Pruning for Accelerating Very Deep Neural Networks," in *Proc. IEEE Int. Conf. on Computer Vision (ICCV)*, 2017, pp. 1389–1397. https://doi.org/10.1109/ICCV.2017.155

[10] M. Molchanov, A. Mallya, D. Tyree, and M. Ashraf, "Importance Estimation for Neural Network Pruning," in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11264–11272. https://doi.org/10.1109/CVPR.2019.01152

[11] C.G. Pachon, D. Ballesteros, and D. Renza, "Senpis: Sequential network pruning by class-wise importance score," *Applied Soft Computing*, vol. 129, p. 109558, 2022. https://doi.org/10.1016/j.asoc.2022.109558

[12] C.G. Pachon, J.O. Pinzon-Arenas, and D. Ballesteros, "Pruning Policy for Image Classification Problems Based on Deep Learning," *Informatics*, vol. 11, no. 3, p. 67, Sep. 2024. https://www.mdpi.com/2227-9709/11/3/67

[13] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the Value of Network Pruning," in *Proc. Int. Conf. on Learning Representations (ICLR)*, 2019. https://arxiv.org/abs/1810.05270

[14] T. Gale, E. Elsen, and S. Hooker, "The State of Sparsity in Deep Neural Networks," *arXiv preprint*, arXiv:1902.09574, Feb. 2019. https://arxiv.org/abs/1902.09574

[15] M. Mondal, S. Das, R. Sarkar, and P. Mitra, "Adaptive CNN filter pruning using global importance metric," *Computer Vision and Image Understanding*, vol. 222, p. 103511, 2022. https://doi.org/10.1016/j.cviu.2022.103511

[16] C. Yang and H. Liu, "Channel pruning based on convolutional neural network sensitivity," *Neurocomputing*, vol. 507, pp. 97–106, 2022. https://doi.org/10.1016/j.neucom.2022.07.079

[17] S. Moon, Y. Byun, J. Park, S. Lee, and Y. Lee, "Memory-Reduced Network Stacking for Edge-Level CNN Architecture with Structured Weight Pruning," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 9, no. 4, pp. 735–746, Dec. 2019. https://ieeexplore.ieee.org/document/8894142

[18] M. Mondal et al., "Adaptive CNN filter pruning using global importance metric," *Computer Vision and Image Understanding*, vol. 222, p. 103511, 2022. https://doi.org/10.1016/j.cviu.2022.103511

[19] C. Yang and H. Liu, "Channel pruning based on convolutional neural network sensitivity," *Neurocomputing*, vol. 507, pp. 97–106, 2022. https://doi.org/10.1016/j.neucom.2022.07.079

[20] Z. Chen et al., "Latency-aware automated pruning with dynamic-based filter selection," *Neural Networks*, vol. 152, pp. 407–418, 2022. https://doi.org/10.1016/j.neunet.2022.04.023