ActNAS : Generating Efficient YOLO Models using Activation NAS

Sudhakar Sah, Ravish Kumar, Darshan C. Ganji, Ehsan Saboori
Deeplite, Inc.
Toronto, Canada
sudhakar@deeplite.ai

Abstract

Activation functions introduce non-linearity into Neural Networks, enabling them to learn complex patterns. Different activation functions vary in speed and accuracy, ranging from faster but less accurate options like ReLU to slower but more accurate functions like SiLU or SELU. Typically, same activation function is used throughout an entire model architecture. In this work, we conduct a comprehensive study on the effects of using mixed activation functions in YOLO-based models, evaluating their impact on latency, memory usage, and accuracy across CPU, NPU, and GPU edge devices. We also propose a novel approach that leverages Neural Architecture Search (NAS) to design YOLO models with optimized mixed activation functions. The best model generated through this method demonstrates a slight improvement in mean Average Precision (mAP) compared to baseline model (SiLU), while it is 22.28% faster and consumes 64.15% less memory on the reference NPU device. Code and data are available at [Link].

1 Introduction

Improving the accuracy of computer vision models has become a highly competitive area, with researchers constantly refining or developing new architectures to achieve state-of-the-art performance. For instance, the recent YOLO10 [24] architecture has set a new benchmark for object detection within the YOLO family [23], outperforming previous models on the popular Microsoft COCO dataset [12]. The current trend toward higher accuracy often involves designing more complex models by increasing the number of learnable parameters or adding performance-enhancing blocks. However, these gains in accuracy typically come at the cost of higher latency. A common practice in deep learning model design is to use a single activation function across all layers. Activation functions introduce non-linearity, enabling models to learn complex patterns from data. Since these functions are applied throughout the architecture, they significantly influence both model latency and accuracy. For example, the Sigmoid Linear Unit (SiLU) [4] offers better accuracy but operates slower than the faster (but less accurate) Rectified Linear Unit (ReLU)[2]. Through our exploration, we observed that employing different activation functions for various layers in the model can improve mean Average Precision (mAP) by 1-2% and reduce latency by 20-30%, compared to the baseline model using SiLU activation across all layers. In this study, we controlled all other factors that could affect latency and accuracy to isolate the impact of mixed activation function design. Additionally, we propose a Neural Architecture Search (NAS) approach to automatically select the optimal activation function for each layer based on given latency and accuracy constraints.
We summarize our contributions as follows:

- We introduce *ActNAS*, a NAS based method to search for the best configuration of the model using mixed activation function for each layer.

38th Workshop on Fine-Tuning in Machine Learning (NeurIPS 2024).

- We summarize the impact of different activation functions on memory and latency on different edge devices.
- We demonstrate that our ActNAS generated models have better mAP and lower latency compared to baseline models trained on COCO dataset and tested on difference edge devices.

2 Related Works

In building computer vision models, it is a common practice to use the same activation function across all layers of the architecture. For example, YOLO models[23], whether in the n, m, s, l, or tiny variants, consistently use the SiLU [4] activation function due to its superior accuracy compared to alternatives like ReLU [21]. However, this higher accuracy often comes at the cost of increased latency, which opens up the possibility of exploring other activation functions such as ReLU[2], LeakyReLU[25], or Hardswish[7]. While YOLO models do allow the option to switch from SiLU to other activations across the entire architecture, there has been no prior research (based on our knowledge) on using mixed activation functions within the same YOLO model. The closest existing work involves designing hybrid activation functions, simplifying blocks in the architecture by applying a different activation function than the main one or mixing activations in image classification models.

Hybrid Activation: In [20], the authors combined softmax[19] and sparsemax[14] in the final activation layer of a Convolutional Neural Network (CNN) for gait analysis using silhouettes. Similarly, in [3], the authors proposed an activation function that can replace ReLU, SiLU, and Hardswish in deep learning models, providing a hybrid approach.

Block Simplification: YOLOv6-3.0 [10] introduced the idea of simplifying the neck of YOLO models [23] by replacing the SiLU activation in the SPPF block [6] with ReLU, creating the SimSPPF block. This was further modified into the SimCSPSPPF block to enhance performance. These works demonstrate that different activation functions have unique strengths and weaknesses, motivating us to explore mixed activations within a single YOLO architecture.

Designing Model Architecture using NAS: In [27], Zoph demonstrated how to design Recurrent Neural Networks (RNNs) with constituent nodes using different activation functions such as ReLU, identity, tanh, or sigmoid. Building on this, [17] introduced a method to search across activation functions like ReLU and Gaussian-smoothed ReLU. Zhenyu [26] further proposed Eigen-NAS, a train-free algorithm that searches for optimal skip connections and activations for each layer in image classification models, showing improvements on datasets like CIFAR-100 and ImageNet-16 but observed a drop in accuracy with CIFAR-10. Our work extends this approach of mixing activations to YOLO architectures by proposing ActNAS, a solution that optimizes activation functions within YOLO models based on layer-specific accuracy and latency data.

Zero-Cost(ZC) Estimators: Zero-cost estimators, which provide accuracy predictions without full training, have been shown to significantly reduce training time. In [9], Ivan et al. analyzed various ZC estimators, including those from [15, 1, 11], in the context of YOLO models, concluding that the NWOT metric [15] performs well for networks using different activations. However, ZC estimators have not yet been applied to assess the micro-level impact of architectural changes, such as altering individual activation functions within a model.

3 Methodology

Unlike the straightforward approach of using the same activation function across all layers in a model, mixed activation models incorporate multiple activation functions within the architecture. As illustrated in Figure 5, the YOLO5n model with SiLU activation achieves the highest mAP but is the slowest across different devices, while the model with ReLU activation is the fastest, though it experiences a significant drop in mAP. The goal of mixed activation models is to identify the optimal combination of activation functions for each layer, creating a model that strikes the best balance between latency and accuracy.

For our experiments, we use the YOLO5n model combined with the YOLO8n head (which offers better accuracy), with the SiLU-based model serving as the reference. As shown in Figure 3, we generate a search space by systematically replacing each activation function in the reference model with a set of candidate activations: *[ReLU, SiLU, Hardswish, ReLU6, and LeakyReLU]*. We replace

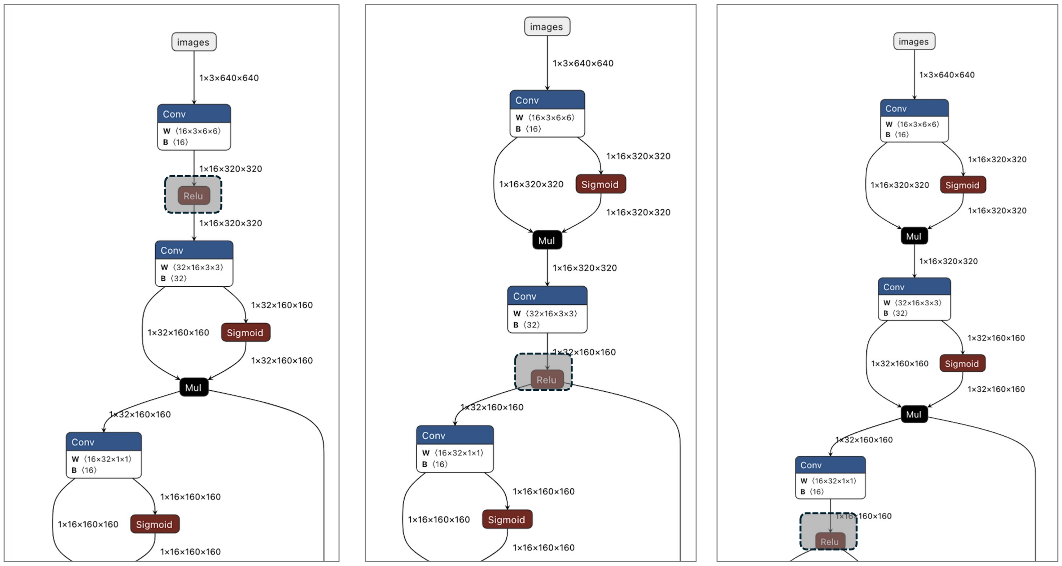

Figure 1: Layer wise activation replacement a) Replace first SiLU activation to ReLU b) Replace second activation c) Replace third activation

one activation at a time, as depicted in Figure 1, resulting in 345 candidate models for YOLO5n (since the reference model has 69 activations and there are 5 candidate activations for each layer).

To evaluate the impact of each activation replacement, we create a new model for each candidate and measure its accuracy, latency, and memory usage, comparing it to the reference model. This process is iterative and is repeated for all 345 candidate models. The results are recorded in performance tables: the latency table, accuracy table, and memory table. Each entry in the accuracy table includes the layer name, activation name, reference accuracy, and delta accuracy (the difference between the reference model's accuracy and the accuracy of the model with the replaced activation). Similarly, the latency and memory tables follow the same structure but record latency and memory values instead of accuracy.

3.1 Accuracy Estimator

As the number of activation and layer combinations increases, the search space for mixed activation models grows exponentially, making it computationally expensive and time-consuming to train each model and evaluate the impact of replacing activations. To address this, we use the NWOT Zero-Cost (ZC) metric [15] to estimate the effect of changing activation functions in individual layers on model accuracy. To validate the NWOT score's relation with the accuracy values for the fully trained YOLO models, we calculated the correlation between the mean-average-precision (mAP) values for all the fully trained models present in MCUBench[22] and their corresponding NWOT scores. The figure 2 shows that the NWOT score correlates strongly with the accuracy of fully trained models. This high correlation indicates that the NWOT score can reliably predict the impact of per-layer activation changes on accuracy without the need for full training. For each candidate activation replacement, the NWOT score is calculated, and the accuracy table is updated accordingly, as described in the previous section. This approach significantly reduces the computational cost and time required to explore the mixed activation model search space.

3.2 Latency Computation

The impact of different activation functions on latency and memory depends on the runtime and hardware architecture. For this reason, a variety of hardware, including two CPUs (ARM Cortex-A57 in Jetson Nano and ARM Cortex-A53 on Raspberry Pi 3), an embedded GPU (Jetson Nano), and a reference Neural Processing Unit (NPU) is used for our experiments. To measure the performance of the models on these devices, ONNX Runtime [18] for Jetson Nano, TensorFlow Lite (TFLite) [13] for the ARM Cortex processors, and a custom compiler/runtime for the NPU is used. Each model

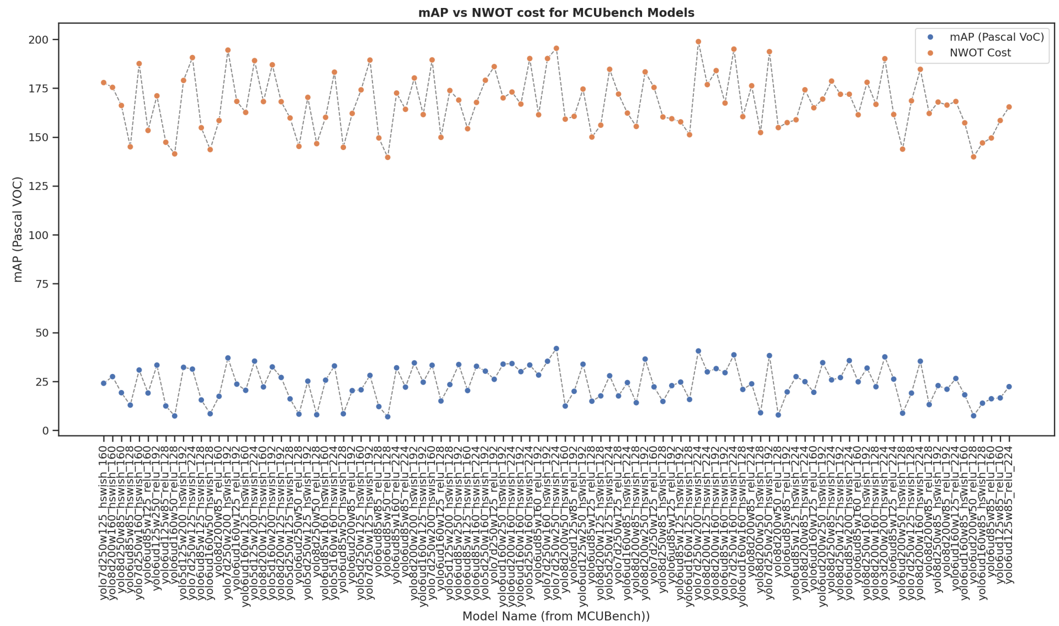

Figure 2: mAP vs NWOT cost for MCUbench[22] Models

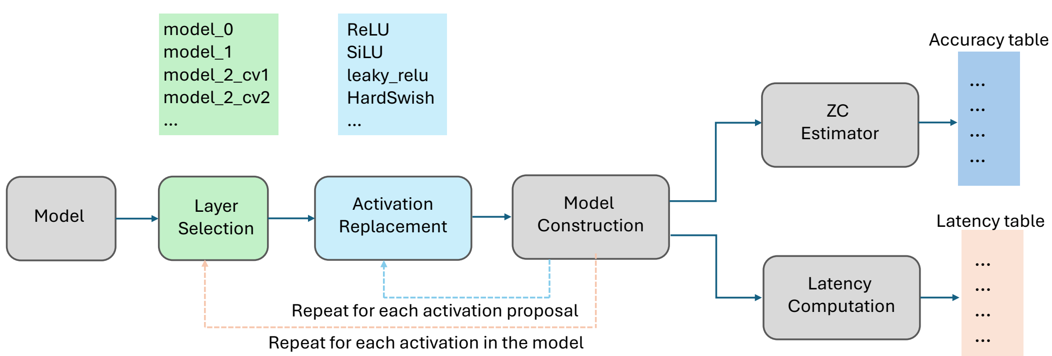

Figure 3: Model Benchmarking using training free estimators and on device inference

is converted to ONNX, TFLite, or a custom format depending on the hardware. Latency values are recorded as average over 50 runs to ensure correctness. The input size of 224x224 is used for NPU due to memory limitations while other latency numbers are calculated using 640x640 input. This variation in hardware and runtime ensures that the results aren't biased towards a specific platform and or runtime, providing a balanced perspective of how different activations perform across different systems.

3.3 Mixed Activation Model Search

Latency, memory and accuracy table obtained from benchmarking step is used to construct mixed activation models.

3.3.1 Local Zero Cost Maxima Approach

Our simple approach to create a model using just two activation functions, such as SiLU and ReLU. Starting with a reference model that uses only SiLU, we apply the Local Zero Cost Maxima (LZCM) method to iteratively replace activations with ReLU based on the zero-cost NWOT score. If the NWOT score improves when a specific layer's SiLU activation is replaced with ReLU, that replacement is kept. The modified model is then trained from scratch on the COCO dataset. Using this simple approach, we developed two models: LZCM1, which mixes SiLU and ReLU, and LZCM2, which mixes Hardswish and ReLU activations.

3.3.2 Naive Approach

Benchmarking models generated using the LZCM approach shows that activations in layers closer to the input have a significant impact on both latency and memory due to the larger feature sizes in those layers. Replacing the activations in the initial layers with ReLU significantly reduces latency and memory usage. In our simple approach, we replace the first three activations near the input with ReLU, while keeping the remaining layers as SiLU. This leads to an efficient balance between performance and resource consumption.

3.3.3 Activation NAS Approach

The LZCM method follows a straightforward decision-making process, replacing activations in individual layers based on their zero-cost score. However, this approach only optimizes each layer locally and doesn't account for the combined impact of replacing activations in multiple layers at once. In contrast, the Activation NAS (ActNAS) approach searches for the optimal activation function for each layer while considering a global budget for latency, memory, or accuracy. The search space for ActNAS is defined by the number of layers multiplied by the number of activation function options. To navigate this space, we use the latency, memory, and accuracy tables, along with predefined constraints, to guide the search. For ActNAS, we experimented with both random search and Integer Linear Programming (ILP) to find the best activation configurations.

Random Search As shown in Figure 4, generating model proposals using Activation NAS requires pre-computed latency and accuracy tables. These tables capture the changes in these metrics when a single layer's activation in the original YOLO model (with SiLU activation) is replaced by an alternative activation. The tables are then converted into latency and accuracy cost matrices, where the rows represent the layers being modified and the columns represent candidate activations (SiLU, ReLU, Hardswish, LeakyReLU, and ReLU6). In addition to these matrices, we also define latency or accuracy constraints, which serve as the overall budget that the algorithm can work within when constructing new models with mixed activations. The random search process begins by randomly selecting activations from the search space that fit within the given constraints. It then computes the total cost (in terms of latency or accuracy) for the model and marks it as the best possible model at that point. The algorithm repeats the random search, comparing each newly constructed model with the previous best. If a new model has a better overall cost, it updates the best model definition and continues iterating through the random selection process.

ILP Search To generate model proposals with mixed activations, we first constructed latency and accuracy matrices, where each element represents the difference in metrics from the original SiLU-based YOLO model when a single layer's activation is replaced. We then formulated this problem as an Integer Linear Programming (ILP) optimization problem. The objective was to minimize the latency and accuracy cost while adhering to constraints on the overall latency or accuracy of the newly constructed models. In the ILP formulation, one variable vector represents the indices of the layers where activations are applied, and the other represents the candidate activations for those layers. We used the open-source PuLP library [16], a Python toolkit for linear programming, to solve this ILP problem. The result is a ranked list of the top k model proposals. To ensure diversity among proposals, the solution avoids excessive overlap, keeping the model architectures meaningfully different.

In Tables 2 and 1, the ActNAS1 model was constructed by applying an accuracy constraint while minimizing latency for the Cortex A-53 CPU hardware. The ILP search can be adapted for different hardware targets by imposing a latency constraint on the constructed model. For example, ActNAS2 and ActNAS3 were built by setting constraints for latency on the Jetson Nano GPU and Cortex A-53 CPU, respectively, with the goal of minimizing accuracy cost while staying within the latency budget for each device.

3.4 Model Fine-tuning

We used Ultralytics [8] to train and fine-tune all our models. For consistency across experiments, we kept all hyperparameters exactly the same, training each model from scratch for 300 epochs using

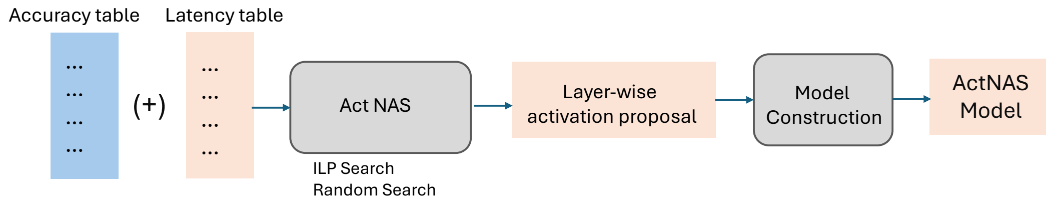

Figure 4: Activation NAS Process

640×640 COCO images and evaluating them on the COCO validation set which contains 5000 images. While mixed activation models could be fine-tuned for a few epochs starting from pre-trained weights, we chose to train all models from scratch to ensure a fair comparison.

Table 1: Performance of ActNAS models compared to baseline models on GPU and CPU

Model	mAP	Jetson Nano			Cortex A-53		
		Latency (ms)	Improvement (%) Hswish	SiLU	Latency (ms)	Improvement (%) Hswish	SiLU
YOLO5n_SiLU	**0.3400**	**27.20**	-	-	937.89	-	-
YOLO5n_ReLU	**0.3205**	**20.18**	-	-	**793.27**	-	-
YOLO5n_Hardswish	**0.3342**	**25.63**	-	-	**823.91**	-	-
LZCM1(SiLU/ReLU)	0.3360	26.61	-21.54%	2.17%	916.97	-0.58%	2.23%
LZCM2(SilU/Hswish)	0.3346	31.15	-21.54%	-14.52%	828.72	-0.58%	11.64%
Naive(ReLU/SiLU)	0.3380	24.55	4.21%	9.74%	906.78	-10.06%	3.32%
ActNAS1(Mixed)	**0.3420**	**24.78**	**3.32%**	**8.90%**	**809.96**	**1.69%**	**13.64%**
ActNAS2(Mixed)	0.3320	23.50	**8.31%**	**13.60%**	966.74	-17.34%	-3.08%
ActNAS3(Mixed)	**0.3400**	**24.51**	**4.37%**	**9.89%**	**965.64**	-17.20%	-2.96%

4 Results

For all experiments in this paper, we used the YOLOv5n model [8]. Initial benchmarking was conducted on the Pascal VOC dataset [5] to avoid training multiple models on the larger COCO dataset [12], with final benchmarking results generated using COCO. Tables 1 and 2 summarize mAP, RAM (NPU only), and latency values for both the reference models and our mixed activation models. The baseline models consist of YOLO5 trained on COCO using SiLU, ReLU, and Hardswish activations, respectively. The LZCM1 model aims to minimize overall latency by mixing SiLU and ReLU activations using a simple search approach, while the LZCM2 model prioritizes maximizing accuracy by combining the more accurate activations, SiLU and Hardswish. Both models achieve

Table 2: Performance of ActNAS models compared to baseline models on NPU

Model	mAP	NPU			NPU		
		Latency (ms)	Improvement (%) Hswish	SiLU	RAM (KB)	Improvement (%) Hswish	SiLU
YOLO5n_SiLU	**0.3400**	**22.35**	-	-	1230.00	-	-
YOLO5n_ReLU	**0.3205**	**17.46**	-	-	**588.00**	-	-
YOLO5n_Hardswish	**0.3342**	**18.53**	-	-	**392.00**	-	-
LZCM1(SiLU/ReLU)	0.3360	21.87	-5.50%	2.15%	1200.00	-206.12%	2.44%
LZCM2(SiLU/Hswish)	0.3346	19.55	-5.50%	12.53%	624.75	-59.38%	49.21%
Naive(SiLU/ReLU)	0.3380	21.43	-15.65%	4.12%	661.50	-68.75%	46.22%
ActNAS1(Mixed)	**0.3420**	**17.37**	**6.26%**	**22.28%**	514.50	-31.25%	**58.17%**
ActNAS2(Mixed)	0.3320	19.27	-3.99%	**13.78%**	520.63	-32.81%	**57.67%**
ActNAS3(Mixed)	**0.3400**	**17.50**	**5.56%**	**21.70%**	**441.00**	-12.50%	**64.15%**

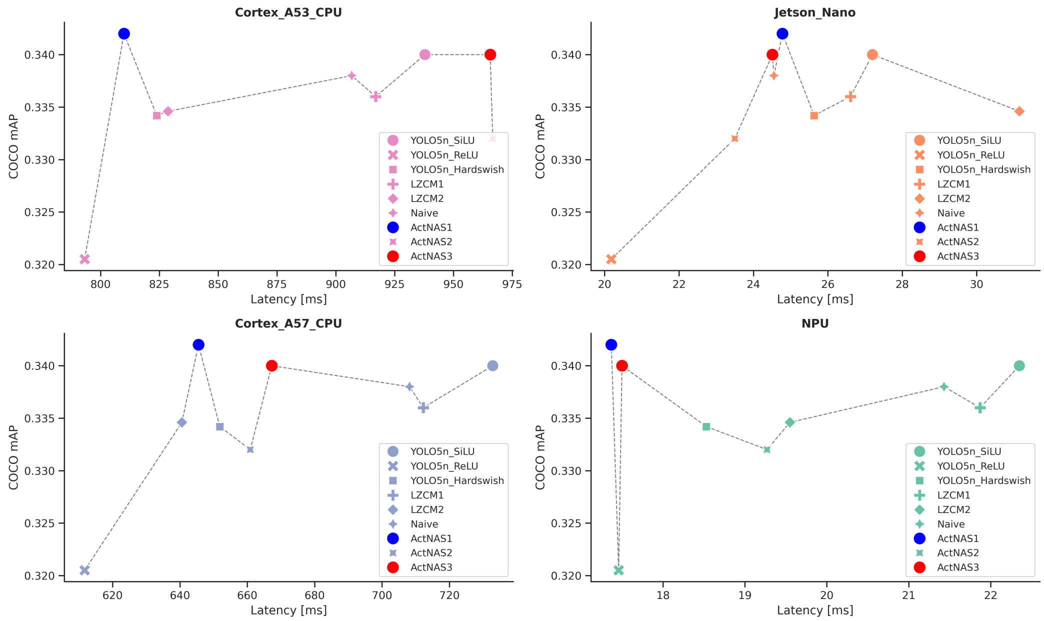

Figure 5: Performance of reference models and activation NAS generated models on different devices.

mAP values slightly lower, but very close to, those of the SiLU and Hardswish reference models. In terms of latency, the LZCM1 model performs slightly better than the SiLU model across all devices (e.g., 21.87 ms vs. 22.35 ms on the NPU). The SiLU/Hardswish mixed activation model demonstrates a better accuracy-latency trade-off, being 12.5% faster than the SiLU model but 5.3% slower than the Hardswish model on the NPU. A similar trend is observed on A53 and A57 CPUs as well as the Jetson Nano GPU. Lastly, the naive approach of replacing the first three activations from SiLU to ReLU resulted in slightly slower performance compared to other baseline models, but the drop in mAP was minimal (0.002) compared to the SiLU model.

ActNAS1 and ActNAS3 were generated using our Activation NAS approach, targeting the NPU as the primary device, as discussed in the previous section. ActNAS1 achieved the highest mAP and the lowest latency, showing 6.26%, 3.32%, 0.97%, and 1.69% improvements in efficiency on the NPU, Jetson Nano GPU, Cortex A57 CPU, and A53 CPU, respectively, compared to the Hardswish model. It also outperformed the SiLU models by 22.28%, 8.90%, 11.92%, and 13.64% on the same devices. Additionally, we found that ActNAS1 required 58% less memory than the SiLU model on the NPU. These experiments demonstrate that the performance of YOLO models can be preserved by creating mixed activation models using a zero-cost proxy-based accuracy estimation, along with latency and memory computation on the target device, paired with an effective NAS approach. ActNAS2, generated with the Jetson Nano GPU as the target, outperformed both the baseline and Hardswish models on Jetson Nano, highlighting the hardware-aware nature of our search method, as it adapts to different hardware profiles. We also observed that the best latency-accuracy trade-off models tended to use ReLU variants (ReLU6, LeakyReLU, and ReLU) in the initial layers. None of the ActNAS models utilized SiLU or Hardswish in the first three layers. This suggests that replacing the initial layers with more efficient activations and following them with high-performance activations—selected via NAS to minimize accuracy loss—produces optimal models across a range of hardware devices. Figure 5 compares our models with baseline models, showing that Activation NAS models offer the best accuracy-latency trade-off on all tested hardware.

5 Conclusion

In this paper, we introduced a novel approach called Activation NAS (or ActNAS), which generates YOLO model architectures using a search space of different activations for individual constituent layers. Our mixed activation models demonstrate slightly better performance than the SiLU-based

YOLO model on the COCO dataset, while being significantly more efficient, with 6% to 23% lower latency depending on the hardware. The NAS approach leverages the zero-cost NWOT score to avoid training a large set of candidate models, making this method practical for model optimization. These experiments suggest that, instead of using a single activation function across the entire model, a combination of latency and accuracy budgets can be used to create custom mixed activation models that fit within specified accuracy, latency, and memory constraints. Future work will expand the benchmarking results to YOLO models with greater width and depth, allowing us to further explore performance improvements and trends in speedup.

References

[1] Abdelfattah, M.S., Mehrotra, A., Dudziak, Ł., Lane, N.D.: Zero-cost proxies for lightweight nas. arXiv preprint arXiv:2101.08134 (2021)

[2] Agarap, A.: Deep learning using rectified linear units (relu). arXiv preprint arXiv:1803.08375 (2018)

[3] Bodyanskiy, Y., Kostiuk, S.: Adaptive hybrid activation function for deep neural networks. System research and information technologies pp. 87–96 (2022)

[4] Elfwing, S., Uchibe, E., Doya, K.: Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. Neural networks **107**, 3–11 (2018)

[5] Everingham, M., Van Gool, L., Williams, C.K.I., Winn, J., Zisserman, A.: The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results (2012), http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html

[6] Glenn, J.: (2022), https://github.com/ultralytics/yolov5/releases/tag/v6

[7] Howard, A., Sandler, M., Chu, G., Chen, L.C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al.: Searching for mobilenetv3. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 1314–1324 (2019)

[8] Jocher, G.: Ultralytics yolov5 (2020). https://doi.org/10.5281/zenodo.3908559, https://github.com/ultralytics/yolov5

[9] Lazarevich, I., Grimaldi, M., Kumar, R., Mitra, S., Khan, S., Sah, S.: Yolobench: Benchmarking efficient object detectors on embedded systems (2023), https://arxiv.org/abs/2307.13901

[10] Li, C., Li, L., Geng, Y., Jiang, H., Cheng, M., Zhang, B., Ke, Z., Xu, X., Chu, X.: Yolov6 v3. 0: A full-scale reloading. arXiv preprint arXiv:2301.05586 (2023)

[11] Li, G., Yang, Y., Bhardwaj, K., Marculescu, R.: Zico: Zero-shot nas via inverse coefficient of variation on gradients. arXiv preprint arXiv:2301.11300 (2023)

[12] Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L., Dollár, P.: Microsoft coco: Common objects in context (2014), http://arxiv.org/abs/1405.0312, cite arxiv:1405.0312Comment: 1) updated annotation pipeline description and figures; 2) added new section describing datasets splits; 3) updated author list

[13] Lite, T.: Tensorflow lite for mobile and edge (2024), https://onnx.ai/, accessed: 2024-07-16

[14] Martins, A., Astudillo, R.: From softmax to sparsemax: A sparse model of attention and multi-label classification. In: International conference on machine learning. pp. 1614–1623. PMLR (2016)

[15] Mellor, J., Turner, J., Storkey, A., Crowley, E.J.: Neural architecture search without training. In: International Conference on Machine Learning. pp. 7588–7598. PMLR (2021)

[16] Mitchell, S., Consulting, S.M., O'Sullivan, M., Dunning, I.: Pulp: A linear programming toolkit for python (2022), https://optimization-online.org/?p=11731

[17] Nguyen, Q.N., Mondelli, M.: Global convergence of deep networks with one wide layer followed by pyramidal topology. Advances in Neural Information Processing Systems **33**, 11961–11972 (2020)

[18] ONNX: Open neural network exchange (2024), `https://onnx.ai/`, accessed: 2024-07-16

[19] Pearce, T., Brintrup, A., Zhu, J.: Understanding softmax confidence and uncertainty. arXiv preprint arXiv:2106.04972 (2021)

[20] Privietha, P., Raj, V.J.: Hybrid activation function in deep learning for gait analysis. In: 2022 International Virtual Conference on Power Engineering Computing and Control: Developments in Electric Vehicles and Energy Sector for Sustainable Future (PECCON). pp. 1–7. IEEE (2022)

[21] Ramachandran, P., Zoph, B., Le, Q.V.: Searching for activation functions. arXiv preprint arXiv:1710.05941 (2017)

[22] Sah, S., Ganji, D.C., Grimaldi, M., Kumar, R., Hoffman, A., Rohmetra, H., Saboori, E.: Mcubench: A benchmark of tiny object detectors on mcus. arXiv preprint arXiv:2409.18866 (2024)

[23] Terven, J., Córdova-Esparza, D.M., Romero-González, J.A.: A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas. Machine Learning and Knowledge Extraction **5**(4), 1680–1716 (2023)

[24] Wang, A., Chen, H., Liu, L.: Yolov10: Real-time end-to-end object detection. arXiv preprint arXiv:2405.14458 (2024)

[25] Xu, B.: Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853 (2015)

[26] Zhu, Z., Liu, F., Chrysos, G., Cevher, V.: Generalization properties of nas under activation and skip connection search. Advances in Neural Information Processing Systems **35**, 23551–23565 (2022)

[27] Zoph, B.: Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578 (2016)