engineering proceedings

MDPI

Proceeding Paper

Evaluating Compact Convolutional Neural Networks for Object Recognition using Sensor Data on Resource-Constrained Devices †

Icaro Camelo [1,*] and Ana-Maria Cretu [1]

[1] Department of Computer Science and Engineering, University of Quebec in Outaouais, Gatineau, QC, Canada
* Correspondence: veri02@uqo.ca (I.C.); ana-maria.cretu@uqo.ca (A-M.C.);
† Presented at the 10th International Electronic Conference on Sensors and Applications, 15–30 November 2023; Available online: https://ecsa-10.sciforum.net/.

Abstract: The goal of this paper is to evaluate various compact CNN architectures for object recognition trained on a small resource-constrained platform, the NVIDIA Jetson Xavier. Rigorous experimentation identifies the best compact CNN models that balance accuracy and speed on embedded IoT devices. The key objectives are to analyze resource usage such as CPU/GPU and RAM used to train models, the performance of the CNNs, identify trade-offs, and find optimized deep learning solutions tailored for training and real-time inference on edge devices with tight resource constraints.

Keywords: machine learning; compact convolutional networks; object recognition resource-constraint devices; IoT; sensor data processing

Citation: Camelo, I.; Cretu, A.-M. Evaluating Compact Convolutional Neural Networks for Object Recognition using Sensor Data on Resource-Constrained Devices. **2023**, *5*, x. https://doi.org/10.3390/xxxxx

Published: 15 November 2023

1. Introduction

Nowadays, artificial intelligence (AI) has become very prominent and impactful owing to its proficiency in accomplishing a wide variety of tasks with high levels of effectiveness and efficiency. Some of the areas where AI has demonstrated its capabilities include, but are not restricted to, visual recognition tasks like image classification, object detection, sensor data and natural language processing. Deep learning is an advanced sub-discipline of machine learning that emphasizes refining artificial neural networks with multiple layers to apprehend intricate representations of data. It can learn useful features from raw data without manual feature engineering. In contrast, the advent of Internet of Things devices having inbuilt sensors opens novel prospects for implementing convolutional neural networks (CNNs) directly on resource-limited devices. However, these devices have limited memory, storage, and computing power, making extensive, complex CNNs infeasible. Implementing compact CNNs with smaller models and computational needs on IoT devices enables localized capabilities like object recognition without relying on the cloud. This reduces latency while improving privacy and reliability. To facilitate model training and inference, several types of specialized hardware have emerged such as CPUs, graphics processing units (GPUs)/ tensor processing units (TPUs), and field-programmable gate arrays (FPGAs).

Researchers have been investigating the training and inference performance of models in resource-constrained devices. Ajit et al. [1] provide a broader review of CNNs without directly addressing the impact of training using different hardware. Nevertheless, the paper offers valuable context regarding the algorithmic steps and applications of CNNs across various fields. Recent studies [2] indicate that both GPU and TPU significantly improved the performance and accuracy of CNN models, with TPU outperforming GPU in certain cases. This suggests that the choice of hardware can have an important impact on model accuracy and overall performance. Other work focuses on GPU and TPU deployment for image classification tasks [3]. Only a few papers explore the use of the NVIDIA Jetson Xavier NX platform for deploying imaging applications. Among these, Jabłoński et al. [4] evaluate the performance of Jetson Xavier NX for real-time image processing for plasma diagnostics. The authors implement several image processing algorithms on the platform and evaluate their performance in terms of speed and accuracy. They found that the platform is able to achieve good performance on the image processing tasks, and that it is well-suited for real-time applications due to its fast-processing speeds. Kortli et al. [5] propose a hybrid model that combines a CNN with a long short-term memory (LSTM) network for lane detection and implement and demonstrate the ability to achieve good performance for this task on the Jetson Xavier NX.

The goal of this paper is to evaluate various compact CNN architectures for object recognition in images trained on the NVIDIA Jetson Xavier NX. The key objectives are to analyze resource usage such as CPU/GPU and RAM used to train models, the performance of the CNNs, identify trade-offs, and find optimized deep learning solutions tailored for training and real-time inferencing on devices with tight resource constraints.

2. Materials and Methods

2.1. NVIDIA Jetson Xavier Platform

The NVIDIA Jetson Xavier NX [6] is a system-on-a-chip (SoC) developed by NVIDIA. It is designed for use in a wide range of applications, including autonomous machines, robotics, and edge computing. The Xavier NX is based on the NVIDIA Volta architecture and features a 6-core Arm Cortex-A57 processor, a 512-core NVIDIA Volta GPU, and a deep learning accelerator (DLA). It is designed to be highly energy efficient and has a small form factor, making it suitable for use in devices with limited space and power resources. The Xavier NX can deliver high performance for a range of tasks, including machine learning, image and video processing, and computer vision. It is targeted at developers and OEMs who are looking to build advanced, high-performance systems for a variety of applications.

2.1.1. Setting up the NVIDIA Xavier NX board

The process of installing NVIDIA SDK Manager and Jetpack [7], flashing an SD card, and installing an SSD drive while changing the root file system (rootfs) to the SSD involves a series of specific procedures.

The first step is to download the NVIDIA SDK Manager from the NVIDIA SDK Manager download page. This step requires one to have an NVIDIA Developer account to access the download. Once the file has been downloaded, the terminal must be opened, and one must navigate to the directory where the file is saved. The permission of the file is then changed to make it executable with a specific command. The SDK Manager is then installed by running a particular command in the terminal.

After the SDK Manager has been installed, it can be executed by typing 'sdkmanager' into the terminal and then logging in with the NVIDIA Developer account credentials. Within the SDK Manager, one must select the appropriate hardware configuration in the 'Target Hardware' section. Then, the desired Jetpack version is selected in the 'SDKs' section. One must then follow the prompts to complete the installation process.

Flashing the SD card is a task handled by the SDK Manager during the Jetpack installation process, requiring the SD card to be connected to the host machine. If a manual flash of the SD card is needed, a tool like Etcher [8] can be utilized. One can download and install Etcher, select the image file they want to flash, select the SD card, and start the flashing process.

The installation of the SSD drive and the changing of the 'rootfs' to point to the SSD first necessitates physically connecting the SSD to the device. After this, the SSD must be formatted, which, in Linux, can be done using a specific command, replacing 'sdX' with the appropriate device id. One is then guided through a series of prompts to create a new partition and format it. After the SSD has been formatted, it is mounted by running a specific command. The contents from the SD card are then copied to the SSD using the 'rsync' command. One then must edit the '/boot/extlinux/extlinux.conf' file on the SD card to point to the SSD, changing 'root=/dev/mmcblk0p1' to 'root=/dev/sdX1'. The device is then rebooted, after which the system should boot from the SSD. It is crucial to remember to replace 'sdX' with the user's SSD drive id and '/dev/mmcblk0p1' with the actual root partition. Additionally, it is of paramount importance that one backs up any vital data before proceeding with these steps and proceeds with caution when modifying system files or disk partitions.

2.1.2. Deploying CNNs on NVIDIA Jetson Xavier NX

Once the above steps are completed, one can begin creating and training machine learning models. Code development can be made more effective by installing the proper Integrated Development Environment (IDE) on the Jetson Xavier NX or by establishing a remote connection over SSH. Pytorch and Tensorflow are both available in the NVIDIA Jetson SDK. The PyTorch library was our choice for implementation. Moreover, we also use Torchvision, which is a PyTorch add-on library that provides datasets, model architectures, and image transformations for computer vision, NumPy [10] for numerical operations, and Scikit Learn [28] that provides utilities for machine learning, including model evaluation metrics. The Pytorch profiler, torch.profiler, is also used for profiling model inference to analyze GPU/CPU usage and memory consumption. For maximizing performance and get the best out of the NVIDIA Xavier NX, we activated all 8 CPUs, enabling its 6 cores, which makes the board to consume 20 Watts of power. NVIDIA provides a script called 'jetson_clocks'. The script is provided by NVIDIA to optimize the board performance through the implementation of static maximum frequency settings for CPU, GPU, and EMC clocks. It is also recommended to activate fans, but we found that the temperatures are not high when the board is managed with the default values.

2.2. Datasets for Experimentation

In this paper, we focus on 2D object recognition in images. We chose two datasets for experimentation. The first one is CIFAR-10 [9], a well-known dataset in computer-vision for object recognition. It contains 60,000 32x32 color images, all of which contain one of the 10 distinct object classes. Each class is comprised of 6000 images, rendering a grand total of 10 unique object classes. The test batch contains 1000 randomly selected from each class. The second one, STL-10 [10] contains 96x96 color images across 10 classes with 500 training and 800 test images per class, totaling 5,000 labelled training images and 8,000 labelled test images. It is commonly employed to benchmark machine learning models and provides 800 test images per class compared to 1,000 for CIFAR-10, a moderate difference. As the STL-10 dataset has fewer labeled training images with higher resolution (32x32 for CIFAR-10 vs. 96x96 for STL-10), we wanted to observe how well models can generalize to different quantity of data and deal with different image sizes.

2.3. Methodology

A series of compact, lightweight CNN architectures, namely AlexNet, ShuffleNet v2, SqueezeNet, ResNet50 and MobileNet v2 are implemented and evaluated on the Jetson Xavier NX platform. The performance is compared when training the algorithms directly, from scratch on the platform or when using a transfer learning process.

2.3.1. Tested Architectures

We have chosen 5 compact architectures for testing: *AlexNet* [11] is comprised of 8 layers with trainable parameters, including 5 convolutional layers paired with max pooling layers, followed by 3 fully connected layers. Each layer uses a ReLU activation function, except for the output layer. It also uses dropout layers, which prevent the model from overfitting. *ShuffleNetV2* [12] is an efficient, lightweight CNN architecture designed for mobile and embedded vision applications with limited computational resources. Its architecture is composed of 50 layers and incorporates two operations: pointwise group convolution and channel shuffle, which significantly reduce computational costs while still preserving accuracy. The architecture of *SqueezeNet* [13] is based on a shuffle operation that enables channel interleaving, reducing the number of computations required by the network. *ResNet* [14] is a pioneering CNN architecture that utilizes residual connections to enable training of very deep networks. Skip connections allow gradients to flow directly to earlier layers. ResNet's key components include residual blocks, stacked together to form the network, and a bottleneck design for deeper versions. This architecture enabled the training of extremely deep networks, from 48 up to 152 layers. In this paper we use Resnet50, which is a ResNet variant that has 50 layers. Finally, *MobileNetV2* [15] uses depthwise and pointwise separable convolutions to reduce parameters and computations needed while incurring a slight decrease in performance. The architecture introduces inverted residual blocks, a modification of the standard residual block found in the ResNet architectures, which allows efficient training on limited computation power. When training these models, we resized and normalized the input image size for each architecture and we shuffled the training datasets. We also applied data augmentation techniques, i.e. cropping and horizontal flipping. We froze the hidden layers to both avoid relearning generic features and improve training performance.

2.3.2. Test Design and Performance Evaluation

In order to train models and assess their performance, we used the two datasets in section 2.2. Each model is trained initially with 10 epochs and the number of epochs is increased to 30, 50, 60, 100, 150, and 200 epochs, for a fixed batch size of 64. The loss function is set as cross-entropy loss, and the optimization algorithm is Stochastic Gradient Descent (SGD) with a learning rate of 0.001 and momentum of 0.9. For monitoring the training process, a script runs in background collecting CPU/GPU/RAM utilization from the board. Also, the loss is printed every 200 batches. The best model is identified by the highest F1-score with less computation cost. However, the time required to train, and accuracy of a model should also be considered depending on the use case. To test our model, we train from scratch directly on the board and also use transfer learning. The latter takes advantage of knowledge previously learned from models trained on large datasets, in our case, the ImageNet dataset [16]. Transfer learning reduces the time required to train a model as it freezes the hidden layers that contain general knowledge (i.e. uses pre-trained weights obtained during learning on ImageNet dataset) and retrains only a limited number of layers, particularly those towards the output layer that contains the specific knowledge of the target task. It achieves a good performance quicker with reduced computation costs. After training, the model's performance is evaluated on the test sets mentioned in section 2.2. The precision, recall, and F1-score for each model are calculated using the Scikit Learn library's functions.

3. Results

Table 1 and 2 summarize the results we obtained on the two datasets using the five tested CNN architectures when trained from scratch and when using the pre-trained weights computed by transfer learning, with the best performance highlighted in bold. On the CIFAR-10 dataset, the AlexNet model trained from scratch improved its F-score from a modest 0.640 after only 10 epochs to an impressive F-score of 0.824 after 50 epochs and finally to an optimal 0.845 after 200 full epochs of training. Using transfer learning, the performance of the model increases significantly with fewer epochs, achieving an

F-score of 0.911 for 100 training epochs. The pre-trained ShuffleNet model achieved an F-score of 0.916 with only 10 epochs. Unlike the model trained from scratch, this model did not see significant improvement as the number of epochs increased and peaked at an F-score of 0.924 for 30 epochs. The SqueezeNet with transfer learning scored very well (an F-score of 0.902) with only 3 hours of training. The pre-trained ResNet50 achieved 0.841 F-score with 30 epochs. On the other hand, MobileNetV2 showed a modest improvement of 0.078 in F-score when using pre-trained weights vs. training from scratch, but in less than half training time.

Table 1. Summary of best model performance on CIFAR-10.

Model.		epochs	Precision	Recall	F-Score	Time to train
AlexNet	Scratch	200	0.846	0.846	0.845	21h46min
	Pre-trained	100	0.912	0.911	0.911	10h47min
ShuffleNet	Scratch	100	0.740	0.741	0.741	13h02min
	Pre-trained	**30**	**0.924**	**0.924**	**0.924**	4h07min
SqueezeNet	Scratch	50	0.767	0.763	0.761	11h40min
	Pre-trained	30	0.902	0.902	0.902	3h
Resnet50	Scratch	100	0.655	0.647	0.649	38h15min
	Pre-trained	30	0.842	0.842	0.841	1h24min
MobileNetV2	Scratch	150	0.750	0.751	0.750	5h
	Pre-trained	100	0.828	0.830	0.828	2h12min

Overall, the pre-trained models achieved an increased average F-score of 13.2% and an average decrease in computational time of 75.8%. For this dataset, the best performance is associated with ShuffleNet and the fastest model to train is Resnet50, but for a decrease in performance of 10.38%. The best compromise between performance and training time seems to be achieved by the pre-trained SqueezeNet, with a decrease of only 2.2% in performance with respect from the best model but for double the time with respect to the fastest model.

As shown in Table 2, on the STL-10 dataset, consistent with the previous dataset, AlexNet is the one achieving the best performance when trained from scratch on the board. The pre-trained AlexNet and ResNet50 models only require very few epochs to achieve excellent results on this dataset. The pre-trained ShuffleNet scored almost twice better than its from scratch version with the same number of epochs. MobileNetV2 from scratch struggled to learn even after 150 epochs. For this dataset, the pre-trained models achieved an increased average F-score of 34.2% and an average decrease in computational time of 73.4%. With an F-score of 0.995, the AlexNet with transfer learning achieves the best performance on this dataset, while the fastest model (one third of the time required by AlexNet) is SqueezeNet for a decrease of 13.5% in performance.

Table 2. Summary of best model performance on STL-10.

Model		Epochs	Precision	Recall	F-Score	Time to train
AlexNet	Scratch	100	0.985	0.984	0.984	1h33min
	Pre-trained	**30**	**0.995**	**0.995**	**0.995**	30min
ShuffleNet	Scratch	100	0.475	0.475	0.474	1h20min
	Pre-trained	100	0.914	0.913	0.913	1h15min
SqueezeNet	Scratch	100	0.613	0.567	0.571	2h30min
	Pre-trained	10	0.862	0.862	0.861	8min
Resnet50	Scratch	200	0.449	0.452	0.464	3h42min
	Pre-trained	10	0.914	0.915	0.914	10min
MobileNetV2	Scratch	150	0.328	0.276	0.257	33min

	Pre-trained	150	0.786	0.781	0.781	32min

4. Discussion and Conclusions

As expected, the pre-trained models performed very well compared to their counterparts trained from scratch (average of 23.7% over the two datasets), as the base model trained on ImageNet is suitable for the task. Also, the pretrained model took less training time (average of 74.6% shorter over the two datasets). The pretrained AlexNet performed better (8.4% improvement) on STL-10 compared to CIFAR-10, with only 30 epochs instead of 100 epochs on CIFAR-10. The same stays true for the counterpart trained from scratch, i.e., a 13.9 % improvement on CIFAR for half the training epochs. This suggests that AlexNet can learn well with fewer images but with higher resolution samples. The pretrained ShuffleNet performed well on CIFAR-10 with only 30 epochs and on STL-10 took more time (100 epochs) to achieve an F-score greater than 0.90. The ShuffleNet model trained from scratch obtained subpar performance (less than 0.5) on STL-10, suggesting that the model struggles with less data. SqueezeNet achieved an F-score of 0.861 with only 10 epochs on the STL-10 dataset and the performance didn't improve with more epochs. However, it achieves an F-score of 0.902 with 30 epochs on CIFAR-10. Similar to ShuffleNet, the SqueezeNet model trained from scratch obtained a low performance (0.57). This implies that SqueezeNet requires more data to increase performance. ResNet50 achieves an F-score of 0.914 with only 10 epochs on the STL-10 dataset but only scores 0.842 with 30 epochs on CIFAR-10. Like AlexNet, ResNet50 does well with few data but with higher resolution images. MobileNetV2 did not show a big discrepancy in terms of F-score across the two datasets, performing slightly better (4% improvement) on the CIFAR-10 dataset with 50 fewer epochs.

As previously mentioned, the best performing model on the CIFAR-10 dataset was the pre-trained ShuffleNet trained on 30 epochs with an F-score of 0.924, whereas the pre-trained AlexNet model achieved an F-score of 0.995 on the STL-10 dataset when trained with 30 epochs. On average, ShuffleNet achieved an F-score of 0.918 over the two datasets, whereas AlexNet achieved 0.953. Even though AlexNet scored slightly better than ShuffleNet, the training time is another determining factor. On average, ShuffleNet only needed 2h41min to achieve good performance on both datasets, while AlexNet took 5h30min, making ShuffleNet the preferred model. Regarding the speed training, the pre-trained ResNet50 model was the fastest to train on CIFAR-10, taking 1h24min to obtain an F-score of 0.841, which is approximately half the time. SqueezeNet, the second fastest model, took 3h to achieve an F-score of 0.902 on CIFAR-10. On STL-10, the pre-trained SqueezeNet was the fastest, showing an F-score of 0.861 in 8 minutes. On the other hand, the pretrained ResNet50 model took only two extra minutes (10 minutes total) to reach an impressive F-score of 0.914.

It is worth mentioning that during the training time, Jetson Xavier NX was overall extremely efficient in terms of resource utilization, using almost 100% of CPU, GPU, and RAM available.

The primary limitations encountered while training CNNs on the NVIDIA Xavier NX board stemmed from the constrained memory resources available. We encountered some challenges while training larger models such as VGG-11 and VGG-19 on CIFAR-10 with the container repeatedly exiting due to out-of-memory errors. After several attempts, we managed to find an appropriate batch size that works for these models. While the Xavier NX platform enables training in many moderate CNN architectures, memory constraints impose clear limits on model and dataset scale versus high-end GPUs or cloud-based accelerators with abundant RAM. In summary, RAM availability represents the primary bottleneck for more advanced deep learning tasks on this embedded hardware. In our future work we will be exploring alternative optimizers and loss functions that could potentially improve convergence speed, model performance, and robustness. Additionally, leveraging hardware-specific libraries such as Nvidia's TensorRT could also improve inference performance on the Xavier NX via strategies tailored to the GPU architecture.

Author Contributions: Conceptualization, I.C. and A-M.C; methodology, I.C. and A-M.C; software, I.C.; validation, I.C, and A.M.C.; formal analysis, I.C..; investigation, I.C.; resources, A-M.C; data curation, I.C.; writing—original draft preparation, I.C.; writing—review and editing, I.C. and A-M.C; visualization, I.C.; supervision, A-M.C; project administration, A-M.C; funding acquisition, A-M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded in part by the NSERC Discovery grant number DDG-2020-00045 and by the NSERC UTILI grant number CREAT-2019-528123.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. A. Ajit, K. Acharya and A. Samanta, "A Review of Convolutional Neural Networks," 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE), Vellore, India, 2020, pp. 1-5.
2. Ravikumar, A., Sriraman, H., Saketh, P. M. S., Lokesh, S., Karanam, A. Effect of neural network structure in accelerating performance and accuracy of a convolutional neural network with gpu/tpu for image analytics. PeerJ Computer Science 8 (2022).
3. Bochkovskiy, A., Wang, C.-Y., and Liao, H.-Y. M. Yolov4 : Optimal speed and accuracy of object detection. arXiv:2004.10934, https://doi.org/10.48550/arXiv.2004.10934 (2004).
4. Jabłoński, B., Makowski, D., Perek, P., Nowakowski, P. N. V., Sitjes, A. P., Jakubowski, M., Gao, Y., and Winter, A. Evaluation of Nvidia Xavier NX platform for real-time image processing for plasma diagnostics. Energies 15, no. 6: 2088.
5. Kortli, Y., Gabsi, S., Jridi, M., Voon, L. F. L. Y., and Atri, M. Hls-based hardware acceleration on the Zynq SoC : A real-time face detection and recognition system. 2022 IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications, SETIT 2022 (2022), 61–64.
6. NVIDIA Jetson Xavier NX. Available online: https://www.nvidia.com/en-sg/autonomous-machines/embedded-systems/jetson-xavier-nx/ (accessed on 27 September 2023).
7. NVIDIA SDK Manager and Jetpack. Available online: https://developer.nvidia.com/embedded/jetpack (accessed on 27 September 2023).
8. Etcher software. Available online: https://etcher.download/about/ (accessed on 27 September 2023).
9. The CIFAR-10 dataset. Available online: https://www.cs.toronto.edu/~kriz/cifar.html (accessed on 19 September 2023).
10. STL-10 dataset. Available online: https://cs.stanford.edu/~acoates/stl10/(accessed on 19 September 2023).
11. Krizhevsky A.; Sutskever I.; Hinton, G.E., 2017. ImageNet classification with deep convolutional neural networks. Commun. ACM 60, 6 (June 2017), 84–90. https://doi.org/10.1145/3065386
12. X. Zhang, X. Zhou, M. Lin and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), Salt Lake City, UT, USA, 2018 pp. 6848-6856.
13. Iandola, F.N.; Han, S.; Moskewicz, M.W.; Ashraf, K.; Dally, W.J.; Keutzer, K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size. ArXiv preprint arXiv:1602.07360 (2016).
14. He, K.; X. Zhang, X.; S. Ren, S.; Sun, J. Deep Residual Learning for Image Recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 770-778, doi: 10.1109/CVPR.2016.90.
15. Sandler, M.; A. Howard, A.; Zhu, M., Zhmoginov, A.; aChen, L. -C. MobileNetV2: Inverted Residuals and Linear Bottlenecks, 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 2018, pp. 4510-4520, doi: 10.1109/CVPR.2018.00474.
16. Russakovsky, O., Deng, J., Su, H. et al. ImageNet Large Scale Visual Recognition Challenge. Int J Comput Vis 115, 211–252 (2015).

Disclaimer/