Deployment of Deep Learning Networks in the Edge using Compression Techniques.

Martín Romero Romero[a,*]

[a]*Centro Singular de Investigación en Tecnoloxías Intelixentes (CiTIUS), Universidade de Santiago de Compostela, Santiago de Compostela, Spain*

***Abstract*—In recent years, deep learning models have grown exponentially in complexity and size, making their deployment on resource-constrained edge devices a significant challenge. This TFM/MSc explores model compression techniques, including quantization, pruning, and knowledge distillation, to optimize deep neural networks for edge computing while maintaining high accuracy. The proposed methods are applied to a ResNet-50 model trained on the CIFAR-10 dataset, evaluating their impact on model size, inference speed, and accuracy. The optimized models are deployed on the ZCU104 FPGA using Vitis AI, demonstrating the feasibility of running efficient deep learning models on hardware with limited computational resources. Experimental results show that a combination of compression techniques can achieve up to 95% model size reduction while preserving or even improving accuracy. These findings highlight the potential of model compression for enabling real-time inference on edge devices, reducing latency, and improving energy efficiency.**

***Index Terms*—Deep Learning, Quantization, Pruning, Knowledge Distillation, Vitis AI, FPGA.**

I. Introduction

In recent years, deep learning models have experienced exponential growth in terms of complexity and learning capacity, driving breakthroughs in areas such as computer vision and natural language processing. As these models have become more accurate and powerful, they have also increased significantly in size, reaching billions of parameters [1] and demanding a level of computation that is generally only available in high-capacity infrastructures such as data centers and cloud servers. Thus, this increase in performance and model rendering capability also brings with it an increase in memory and processing power consumption [2], which is unsustainable for edge devices such as smart cameras, Internet of Things (IoT) sensors, cell phones and other embedded systems.

In these environments, the inference in deep learning models needs to be performed in real-time or near real-time, while keeping power consumption low to extend device autonomy. At the same time, many of these devices lack deep learning-specific accelerators, which limits their ability to run compute-intensive operations at the same speed as in cloud computing environments. Hardware limitations on edge devices call for minimizing the memory and processing cycles required for inference [3], while ensuring that the model accuracy remains within acceptable margins for each application.

In this context, model compression techniques have become a fundamental tool to make the implementation of artificial intelligence at the edge feasible. These techniques make it possible to reduce both the size of the model and the computational resources required without significantly compromising accuracy and performance. The most commonly used compression techniques include quantization [4], which reduces the accuracy of model parameters; pruning [5], which removes nonessential connections and parameters; and knowledge distillation [6], which transfers learning from a more complex model to a smaller one. Each of these techniques offers a way to optimize the model to fit the constraints of the edge devices.

In addition, the use of compression is crucial to address another growing challenge: latency in data transmission [7]. In edge applications, models must not only run on limited devices, but also reduce dependence on constant communication with a server or cloud, due to the latency involved and the associated risks to data privacy and security. Compression allows models to process more information locally, which improves autonomy and real-time response in critical scenarios.

The goal of this TFM/MSc is to deploy deep learning networks on an FPGA platform, a relatively low-power device ready for their use on the edge. This work is outlined as follows. Section II describes the related work summarizing the main compression techniques used in this project. Section III details the implementation, including the tools, datasets, and hardware resources employed, as well as the workflow for training, compressing, and deploying the models. Section IV presents the results obtained from the experiments, analyzing the impact of the compression techniques on model accuracy, size, and efficiency. Finally, Section V provides the conclusions of this work, while Section VI discusses potential future work and improvements.

II. Related Work

As stated above, the increasing complexity of deep learning models has led to the development of various compression techniques to make them more efficient and suitable for deployment on edge devices, where memory, computation and energy resources are limited. In particular, quantization, pruning and knowledge distillation have proven to be effective optimization methods. A discussion of these techniques is presented below.

*Work supervised by Víctor Brea and Fernando Pardo.
Email addresses: martin.romero.romero@rai.usc.es (Martín Romero Romero), victor.brea@usc.es (Víctor Brea), fernando.pardo@usc.es (Fernando Pardo).

A. Quantization

Neural network quantization has become a fundamental tool for reducing the size of models and their processing demands, enabling their deployment on edge hardware with accuracy and computational capacity limitations. According to [4], quantization reduces the precision of model weights and activations, moving from 32-bit floating-point representations to lower precision representations, such as 8, 4 or even 2 bits, and, on occasions only one bit, without significantly compromising model accuracy. The quantization process can be carried out in several ways:

1) **Post-training quantization (PTQ)**: this method applies quantization after the complete training of the model in floating point. It is an efficient technique in terms of time and resources, since it does not require modifications to the training process. However, PTQ may not preserve the performance of very complex models or models sensitive to accuracy changes as stated in [4], where the authors emphasize that PTQ is suitable for applications with low accuracy-sensitive, such as convolutional image classification networks.
2) **Quantization Aware Training (QAT)**: in this approach, quantization is carried out during training, which allows the model to learn the effects of the quantization. Although it consumes more computational resources, as a new training process must be carried out, QAT usually provides better accuracy results in complex models, since it adjusts the weights to counteract the loss of information during quantization. According to [4], QAT is especially useful for accuracy-sensitive applications such as object detection and multi-class classification.

In [4], it is also highlighted the importance of per-channel and per-tensor quantization. While per-channel quantization allows finer tuning tailored to each filter by adding a scale factor to each channel of the tensor, per-tensor is faster and more direct, applying the same scale to the entire layer. The choice of one or the other depends on the accuracy and speed requirements of the deployment device.

A recent approach to quantization is proposed in [8], where a fast stochastic algorithm for quantizing trained neural networks, based on a greedy tracking mechanism and a stochastic quantizer, is presented. In addition, the algorithm scales linearly with the number of weights, allowing to quantize large networks efficiently.

B. Pruning

Pruning reduces the complexity of deep learning models by eliminating less important parameters and connections. In [9] three categories are introduced for pruning: unstructured, structured, and semi-structured, each with specific advantages and disadvantages.

1) **Unstructured Pruning**: unstructured pruning, or weight pruning, removes individual weights based on their low magnitude or minor impact on accuracy. Although this method achieves high compression and sparsity, its sparse structure is difficult to accelerate on hardware that is not optimized to handle sparse matrices. Thus, although it reduces memory, it does not always improve inference speed without specific hardware.
2) **Structured Pruning**: structured pruning removes entire blocks, such as filters, channels or layers, while maintaining the regular structure of the network. This facilitates a further reduction in inference time and is more compatible with general hardware, such as GPUs and FPGAs. Although it may reduce accuracy somewhat by removing entire components, its structural consistency makes it ideal for edge device applications.
3) **Semi-structured Pruning**: semi-structured pruning, or pattern-based pruning, combines the consistency of structured pruning with the flexibility of unstructured pruning. This method removes groups of weights following specific patterns within filters or channels, allowing for improved efficiency without loss of accuracy. It is especially useful in vision applications, where precision and structural patterning are required for hardware optimization.

A recent approach on pruning techniques is presented in [10], where the authors propose Structured Pruning Adapters (SPAs). These adapters combine structured pruning and efficient parameterization to speed up and specialize networks without losing accuracy. They propose SPLoRA and SPPaRA, which improve performance in image recognition tasks with fewer parameters and pruned models.

C. Knowledge Distillation

Knowledge distillation is a model compression technique in which knowledge from a large, complex model (teacher model) is transferred to a smaller, more efficient model (student model). This technique, introduced in [6], allows the student model to learn both the final predictions of the teacher model and the intermediate features of its reasoning process, thus achieving high accuracy in a smaller model.

Knowledge distillation is based on the use of logits, which are the smoothed probabilities of the outputs of the teacher model. Instead of training the student model exclusively with the original labels of the dataset, a combination of these labels and the logits outputs of the teacher model is used. By mimicking these smoothed outputs, the student model can capture more complex patterns and achieve greater generalization. There are different distillation approaches:

1) **Logit-based distillation**: this method trains the student model to replicate the probability outputs of the teacher model rather than just the class labels. This allows the student model to capture the relationships between classes and subtle patterns that the teacher model has learned, resulting in more robust learning [6].
2) **Intermediate layer distillation**: instead of learning only from the final output of the teacher model, the student model can also mimic the teacher's inner layer representations. This allows the student to emulate the teacher's feature extraction process, improving its performance on tasks where intermediate features play a key role [11].
3) **Progressive distillation and self-learning**: some modern methods combine knowledge distillation with self-

learning, where the student model first trains independently and then iteratively adjusts to improve in areas where the teacher shows better performance [12].

Knowledge distillation is especially useful for creating models that are compact and accurate, making it ideal for applications on edge devices. This method allows the performance of a complex model to be maintained in a compressed format, thus achieving a balance between efficiency and accuracy.

In recent work such as [13], the authors propose a unification of knowledge distillation (KD) and self-distillation (self-KD) using Normalized KD (NKD) and Universal Self-Knowledge Distillation (USKD). NKD normalizes non-target class logits to improve distillation, while USKD generates custom soft labels without the need for a master, applicable to both CNN [14] and ViT [15]. It achieves state-of-the-art results on CIFAR-100 [16] and ImageNet [17], improving accuracy with minimal additional costs.

D. FPGA Implementation

The deployment of deep learning models on edge devices requires not only compression techniques, but also specialized hardware to run these models efficiently. Field-Programmable Gate Arrays (FPGAs) [18] are a type of chip that can be programmed to perform specific hardware tasks and have a flexible architecture with logic blocks that can be configured to perform different types of functions, such as signal processing, system control or algorithm acceleration.

FPGAs have established themselves as a leading solution due to their ability to combine performance, flexibility and power efficiency [19]. Unlike general-purpose processors, FPGAs offer the possibility to customize their hardware to meet the specific needs of neural network inference, such as low latency and high throughput, essential features in edge applications.

For this TFM/MSc, the ZCU104 development board [20] has been chosen as the deployment platform. This board is populated with the Zynq UltraScale+ XCZU7EV-2FFVC1156 MPSoC, which combines a powerful processing system (PS) and programmable logic (PL) on the same device. The PS in a Zynq UltraScale+ MPSoC features the Arm ® flagship Cortex ® -A53 64-bit quad-core processor and Cortex-R5 dual-core real-time processor. All this features make the ZCU104 a powerful and versatile environment for neural network acceleration. The ZCU104 platform has been chosen not only for its computational capabilities, but also for its availability. This board was accessible for the development of this project without the need for additional hardware acquisitions, which simplified the process and reduced costs. Figure 1 shows a photo of said ZCU104 FPGA platform with its dimensions. The ZCU104 FPGA platform is programmed through an Ethernet RJ45 connector with a personal computer without demanding requirements. Additionally, it includes a USB port, which can also be used for connectivity.

The implementation of the compressed models on the ZCU104 was performed using *Vitis AI*, Xilinx's development platform for artificial intelligence inference on FPGAs [21]. *Vitis AI* provides a comprehensive set of tools designed to optimize and compile pre-trained models, adapting them to run efficiently on the FPGA's programmable logic. Key features of *Vitis AI* include:

Figure 1: *ZCU104 board with dimensions 15 cm × 18 cm.*

1) **Model optimization and quantization**: *Vitis AI* allows transforming pre-trained models into lighter versions adapted to the FPGA architecture, maximizing performance while minimizing resource usage.
2) **FPGA-specific compilation**: The tools generate specific hardware implementations that take full advantage of the ZCU104 capabilities.
3) **Acceleration libraries**: *Vitis AI* includes libraries optimized for common neural network operations, facilitating the integration of compressed models into real applications.

The combined use of the ZCU104 and *Vitis AI*, as explored in [22] or [23], allows the evaluation of compression techniques not only from a theoretical perspective, but also in the context of a real deployment. This approach ensures that the results obtained in this work are applicable to real scenarios where edge devices face tight resource constraints and must maintain high performance. Furthermore, the use of this platform illustrates how FPGAs can be used to address the challenges of implementing artificial intelligence models in edge environments, as explored in [24] or [25], opening up new opportunities in industrial and research applications.

III. Methodology

The implementation of this work consists of the development, compression and deployment of deep neural network models using various optimization techniques on the ZCU104 FPGA platform. The main aspects of the implementation are described below, including the frameworks used, the datasets employed, the development process and the hardware resources used.

A. Frameworks

For the implementation of the models and the application of the compression techniques, the following framework has been mainly used:

- **PyTorch**: PyTorch [26] has been the main framework for the development of all the code in this work for the design, training and evaluation of deep neural network models, including quantization, pruning and knowledge distillation techniques. Quantization and pruning were performed both in PyTorch and with Vitis AI to optimize the models for deployment on the ZCU104 FPGA platform. On the other hand, knowledge distillation was implemented entirely within PyTorch, where smoothed logits were used to transfer knowledge from the teacher model to the student model. In addition, PyTorch is known for its easy integration with other libraries and its strong support for GPU operations, which allows for faster and more efficient training and evaluation.
- **Vitis AI**: for the optimization and deployment of the models on the ZCU104, Vitis AI, Xilinx's platform for artificial intelligence inference acceleration on FPGAs, was used. As described in *Section II-D* , Vitis AI facilitates model quantization, optimization and compilation, adapting them to the FPGA hardware to ensure that the compressed models run efficiently.

B. Datasets

The neural network model was trained and evaluated using two datasets, **MNIST** [27] and **CIFAR-10** [28].
MNIST, Figure 2, contains 70,000 grayscale images of 28×28 pixels representing handwritten digits from 0 to 9. It is one of the most widely used datasets for classification and has been used for the first quantization and pruning tests, which have been done on simple handcrafted neural network models.

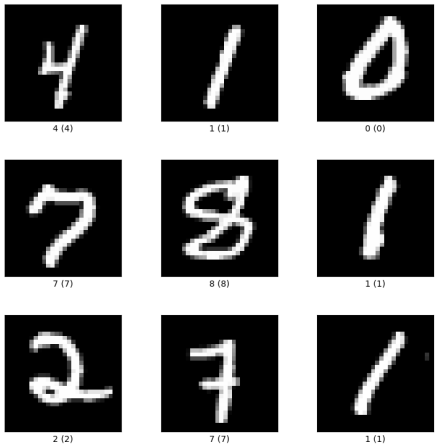

Figure 2: *Samples of the MNIST dataset* [27].

For tests with more complex models, **CIFAR10** has been used, which consists of 60,000 32×32 pixel images distributed in 10 different classes, such as animals, vehicles and objects, as shown in Figure 3. CIFAR-10 is a classic benchmark in the field of computer vision and has been widely used to evaluate the performance of image classification models. This dataset was chosen because of its moderate size and its ability to represent a variety of classes that allow compression techniques to be evaluated efficiently. For model training, the images were resized to 224×224 pixels to best fit the model used, of which 50,000 were used for training and 10,000 for testing.

Figure 3: *Samples of the CIFAR10 dataset* [28].

C. Implementation Process

The implementation process of this work was designed to cover all the necessary stages, from the training of the models to their deployment on specialized hardware. This includes not only the development and compression of the models in PyTorch, but also their optimization to run on the ZCU104 FPGA platform with Vitis AI. At each stage, an attempt was made to ensure that the models maintain a balance between accuracy and computational efficiency after the application of the corresponding compression techniques. The key stages of the implementation process on the FPGA are described below:

1) **Model Training**: initially, a simple neural network model with two fully connected layers of 64 neurons has been trained in the MNIST datasets. This model was designed to pretest and validate the training and evaluation workflow in a controlled environment. However, due to the simplicity of the architecture and dataset, it was not considered necessary to deploy it on the FPGA. Therefore, it was decided to train a more complex model, specifically a ResNet-50 architecture [29], using the CIFAR-10 dataset, which presents a greater variety of classes and complexity in the images. For the training of the ResNet-50, the PyTorch framework was used, implementing standard optimization techniques, such as Adam, to adjust the model parameters and ensure efficient convergence. The model was trained for 10 epochs with a learning rate of 0.001 and using as loss, CrossEntropyLoss.
2) **Application of Compression Techniques**: once the models were trained, different compression techniques were applied to optimize their size and efficiency.
    a) **Post-Training Quantization (PTQ) and Quantization Aware Training (QAT)**: these techniques were used to reduce the size of the model by transforming weights and activations into integer values, adjusting it to operate on resource-constrained hardware environments without significantly compromising its performance. These quantization techniques were implemented using

both PyTorch and Vitis AI, taking advantage of the capabilities of both frameworks.

b) **Structured Pruning**: structured pruning techniques were applied (using Vitis AI), eliminating entire components such as filters and channels to reduce the number of parameters and improve computational efficiency.

c) **Knowledge Distillation**: logit-based and intermediate layer knowledge distillation were performed to transfer learning from a teacher model to a lighter student model. This was implemented exclusively using PyTorch. In the distillation process, different student models were used, ranging from models trained from scratch with convolutional and fully connected layers, to pretrained models such as ResNet-18, SqueezeNet [30], and MobileNet [31].

These tools were used in a complementary manner to maximize the efficiency of the compressed model on specific tasks.

3) **Compilation with Vitis AI**: once the compression techniques were applied, the optimized models were prepared for deployment on the ZCU104 FPGA using Vitis AI tools. This process involved the compilation of the models in a format compatible with the FPGA hardware, ensuring their correct execution on the device.

4) **Deployment on the ZCU104**: finally, the compressed and optimized models were deployed on the ZCU104, using Vitis AI to load and run the inferences on the device. This process involved tweaking the Vitis AI code and acceleration libraries to ensure that the models ran efficiently on the FPGA.

It is important to note that compression techniques, such as quantization, pruning and knowledge distillation, can be used either together or independently, depending on the specific optimization objectives. However, for model compilation and deployment on the ZCU104 using Vitis AI, quantization is a mandatory step. This is caused because the Deep Processing Unit (DPU) implemented on the FPGA is designed to operate with 8-bit integers [32], ensuring that the hardware takes full advantage of its acceleration capabilities and minimizes resource consumption. Therefore, even in combinations with other techniques, quantization is an essential component in the workflow. Figure 4 shows a workflow to illustrate the process of implementing a model from the base model to its final deployment on the FPGA, with all the possible compression techniques that have been introduced in this work, both individually and in combination, as well as the order in which to execute each technique.

The flow of Figure 4 starts with the base model, on which optional steps such as Knowledge Distillation, which is performed exclusively with PyTorch to create a smaller model from the knowledge of the base model, or Pruning, which can be performed with both PyTorch and Vitis-AI to remove redundant weights and reduce the complexity of the model, can be applied. Subsequently, the mandatory quantization step is performed, which reduces the numerical accuracy of the parameters to optimize the model on specialized hardware devices; this step is performed exclusively with Vitis-AI. Finally, the quantized model is compiled with Vitis-AI, adapting it to the format required for the ZCU104, and deployed on the board for execution. This workflow allows the steps to be adapted according to the needs of the project, as optional techniques can be combined or omitted without compromising the final quantization and deployment stage. The combination of the different techniques can be done by following the different dashed lines represented in the figure.

D. Hardware Resources

To conduct the experiments and train the deep learning models, the following hardware resources were used:

- **NVIDIA GTX 1650 MAX Q GPU**: Initially, an NVIDIA GTX 1650 was used, which is the graphics card of a personal computer. This GPU (with 896 CUDA Cores, a memory of 4GB GDDR5 and bandwidth of 128 GB/s) allowed initial testing and training of simpler models, such as the neural network with two hidden layers using the MNIST dataset. However, due to its limitations in memory and computational capacity, it was not suitable for training more complex models, such as the ResNet-50, efficiently.
- **NVIDIA Tesla P40 GPU**: For the training of more complex models and the experimentation with the CIFAR-10 dataset, a Tesla P40 was used, available in one of the servers of *CITIUS* (Centro Singular de Investigación en Tecnoloxías Intelixentes of Universidade de Santiago de Compostela). This GPU, with higher computational power and memory (3840 CUDA Cores, 24 GB GDDR5X of memory and 346 GB/s of bandwidth), allowed training the ResNet-50 and running advanced model compression experiments easily and in a reasonable time.
- **ZCU104**: The ZCU104 board was used for the final deployment of the compressed models. As explained in Section II-D, this board contains a Zynq UltraScale+ MPSoC, which integrates an ARM Cortex-A53 processor with programmable logic (FPGA). The ZCU104 was selected due to its availability in the lab and its powerful processing capabilities, making it an ideal choice for running optimized models in an edge environment.

IV. EXPERIMENTATION

This section presents and analyzes the results obtained after the application of the compression techniques studied in this work: quantization, pruning and knowledge distillation. The experiments were performed both with a simple model trained on the MNIST dataset and with a ResNet-50 trained on CIFAR-10. The included graphs and tables summarize the effects of these techniques in terms of accuracy and model size on the ZCU104 FPGA through VitisAI.

A. Quantization Experimentation

In this section we analyze the results obtained from applying the quantization techniques described in Section II-A. For a

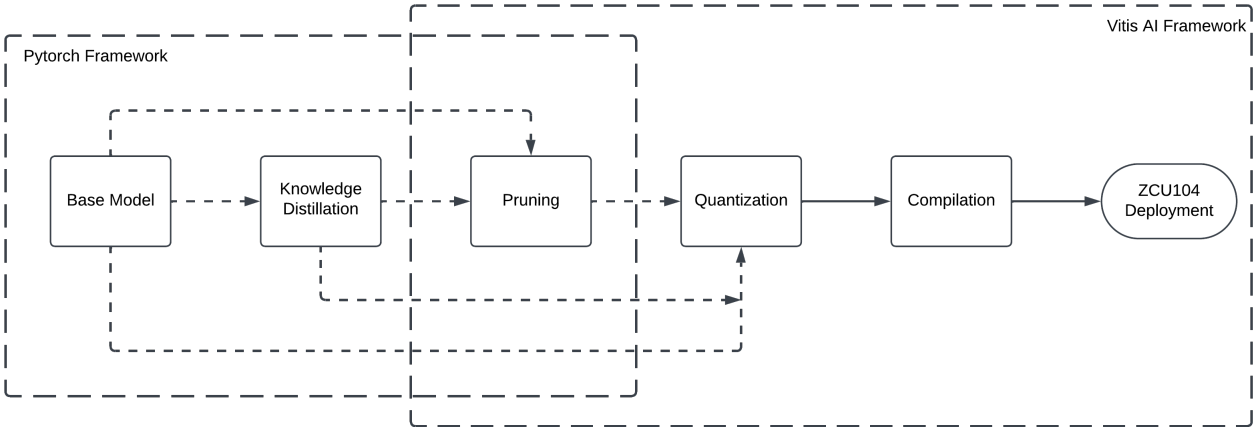

Figure 4: *Workflow from deep learning model PyTorch to ZCU104 FPGA board implementation.*

first test, it was decided to explore quantization in a simple model as preliminary results for a better understanding of how Deep Neural Networks (DNN) compression solutions work. This simple model had 2 fully connected hidden layers, with 64 neurons per layer and a softmax output. On the base model trained with the MNIST dataset, Post Training Quantization was performed with an accuracy of 16 floating bits, 8 integer bits and 5 integer bits; and Quantization Aware Training with an accuracy of 8 and 5 integer bits. As can be seen in Figure 5, the accuracy of all quantized models is not affected compared to the base model. In fact, the accuracy increases in all quantized models except in the *PTQ - Int5* model, but even so, the loss of accuracy with respect to the base model is very small.

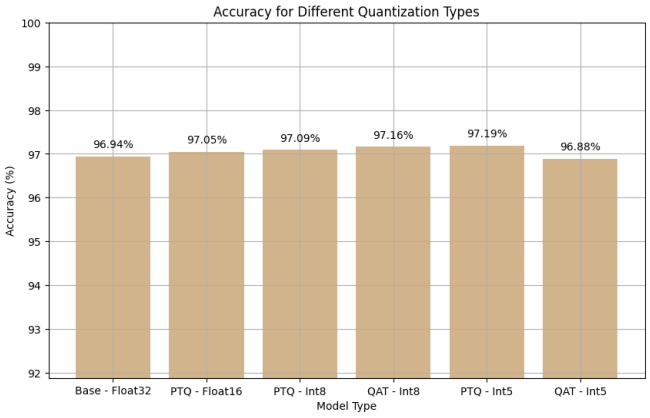

Figure 5: *Accuracy of a test base model of two fully connected hidden layers and its quantized versions (PyTorch).*

Furthermore, when studying Figure 6, which represents the size of the previous models, we see that all the quantized models have been downsized by a $4\times$ factor, which added to the accuracy shown in Figure 5, makes the new quantized models better than the baseline architecture, not only maintaining but even surpassing the accuracy of the baseline solution. The reason for that can be attributed to several factors. Quantization acts as a form of implicit regularization by reducing the precision of weights and activations, which helps prevent overfitting and improves generalization. Additionally, the rounding process during quantization eliminates small variations in the weights, leading to more consistent predictions. Furthermore, the simplicity of the MNIST dataset amplifies these effects, allowing the quantized models to even outperform the baseline accuracy. It is also noteworthy that 5-bit quantized models obtain similar sizes to 8-bit quantized models. This is due to the fact that storage and processing systems are usually designed and optimized to work with multiples of 8 bits (1 byte), so if we have a model with a precision lower than 8 bits, when saving the model, the weights are padded with 0 until having a precision of 8 bits.

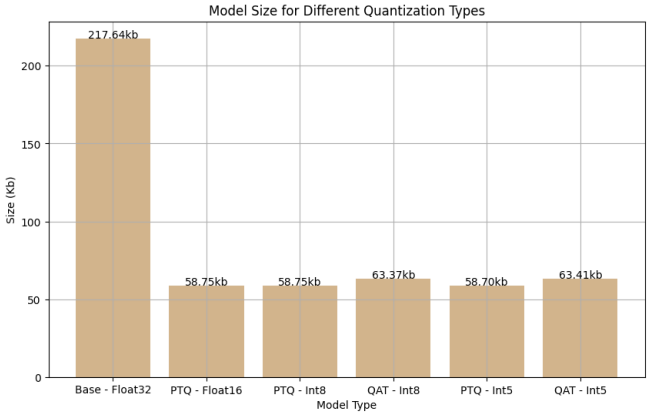

Figure 6: *Size of the test baseline model of 2 hidden layers and its quantized versions (PyTorch).*

Once the results provided by quantization with a simple model of 2 hidden layers has been analyzed, we test if the same conclusions hold for realistic DNN models. The widely known ResNet-50 architecture has been chosen as testbench to this end. Once the Resnet base model was trained, Post-Training Quantization with an accuracy of 8 and 5 integer bits and Quantization Aware Training with an accuracy of 8 and 5 integer bits were performed.

Results for the accuracy and size of the models were obtained once the different quantizations have been carried out, in order to see the extent of the quantization. Figure 7 shows the accuracy of the quantized models and the base

model. If we compare the results, we see that the models quantized with PTQ feature a dramatic loss of accuracy with respect to the base model. However, the two models quantized with QAT have maintained accuracy and have even been able to slightly improve the performance of the base model. The slight improvement in accuracy observed in QAT quantized models can be attributed to the adaptive nature of the QAT process, which allows the model to adjust to the effects of reduced precision. Also, the implicit regularization of the Quantization process and the robustness of the ResNet-50 architecture contribute to maintain the accuracy results.

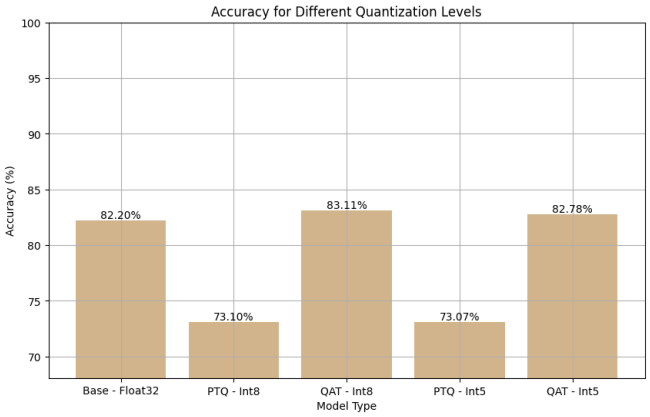

Figure 7: *Accuracy of a ResNet-50 model used as baseline and its quantized versions (PyTorch).*

In Figure 8, the size of each quantized model and its ResNet-50 baseline is shown. We can clearly see that all quantized models are significantly reduced in size with respect to the baseline model. In this way, we have obtained models much smaller than the base model, and in the cases of the QAT models, even improving the accuracy of the base model.

These results confirm that the same conclusions drawn from the simple network also hold for the ResNet-50 architecture. Quantization, especially QAT, not only reduces the model size but can also slightly improve its accuracy, demonstrating its effectiveness as a compression technique for both simple and complex networks.

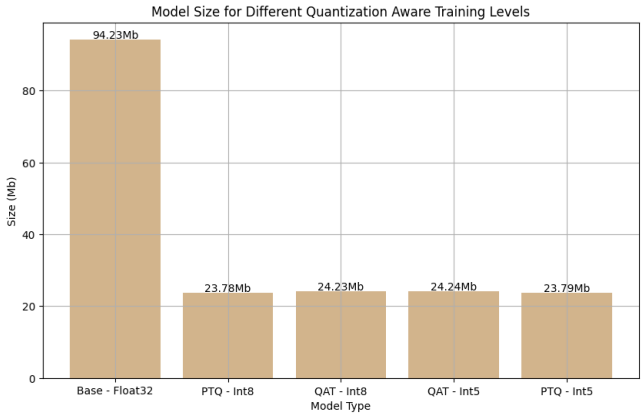

Figure 8: *Size of a ResNet-50 model used as baseline and its quantized versions (PyTorch).*

In the previous quantization experiments, tests were carried out with 8 and 5 integer bit accuracies using only the Pytorch framework, observing a significant reduction in the size of the models. We go one step further by running compression techniques with Vitis AI for the deployment of the ResNet-50 on the ZCU104, and only 8 integer bit quantization was applied. This is because the DPU (Deep Processing Unit) implemented on the FPGA operates with INT8, which prevents taking advantage of the additional size reduction offered by 5-bit quantization. Although the model can be quantized to lower precision (5 bits) in theory, the Vitis AI output format does not allow representing compressed models with a precision lower than 8 bits, keeping the final size of the model unchanged. For this reason, evaluations in Vitis AI focused only on models quantized to 8 bits, ensuring compatibility with the compilation workflow and the deployment on the ZCU104 FPGA.

The results obtained from the PTQ and QAT models trained with Vitis AI are presented below. As mentioned before the models have an accuracy of 8 bits, and, in addition, two models have been trained using QAT.

The first one, *QAT scratch*, has been trained from scratch without pretrained weights, and *QAT weights* has been trained from the weights of the original ResNet-50 model. In Figure 9, the accuracy results of the models are presented. We can see that Vitis-AI is not able to match the accuracy of the original ResNet-50 model, being, as expected, the PTQ model the one with the worst accuracy. It is also surprising that the *QAT scratch* model obtains better accuracy than *QAT weights*, since a priori one would assume that starting the training with the weights of the base model would give an advantage over the model starting with the randomly initiated weights.

One possible explanation for this counterintuitive result is that starting the training from scratch enables the model to adapt more effectively to the constraints set by the quantization process. When initializing with pretrained weights, the model may inherit dependencies on higher-precision representations, complicating its adjustment to the reduced precision during QAT. In contrast, the *QAT scratch* model learns to operate within these constraints from the beginning of the training, which may result in a better alignment with the quantized representation.

It is also important to highlight that it is not possible to deploy a PyTorch-quantized model directly on the ZCU104 without first performing quantization with Vitis AI. This limitation exists because Vitis AI is required to convert the model into a format compatible with the FPGA. Additionally, Vitis AI's quantization process, based on [33] and [34], is not as optimized as PyTorch's, which could explain the observed drop in accuracy compared to the models quantized solely with PyTorch.

Figure 10 shows the size of the quantized models with respect to the base model. As in the other previous cases, we see that the quantized models manage to reduce their size considerably with respect to the base model. So, again we have obtained models significantly smaller than the original, and in the case of the *QAT scratch* model with an accuracy quite close to the base model.

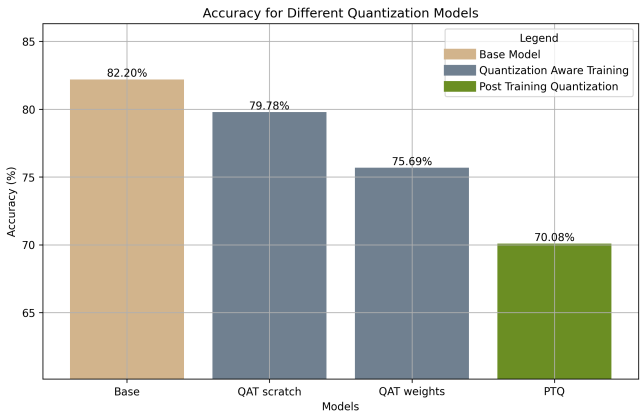

Figure 9: *Accuracy for the original ResNet-50 model and different quantization models with VitisAI.*

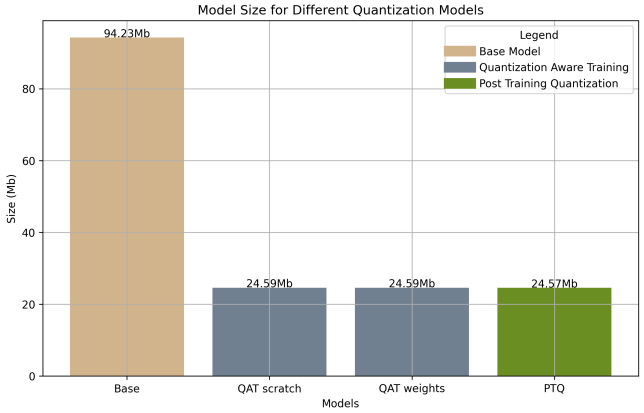

Figure 10: *Size of the original ResNet-50 model and different quantization models with VitisAI.*

B. Pruning Experimentation

This section analyzes the results obtained after applying different degrees of structured pruning on the ResNet-50 model using Vitis AI. The main objective of pruning is to reduce the number of model parameters by eliminating less relevant connections or components, in order to reduce its size and improve its computational efficiency, while maintaining an acceptable level of accuracy. For this analysis, experiments were performed with two structured pruning methods (*iterative* [35] and *one-step* [36]) provided by Vitis AI and with different levels of pruning, evaluating how the progressive elimination of parameters affects both the accuracy and the size of the model. The iterative pruning algorithm reduces model parameters while minimizing accuracy loss by pruning and fine-tuning in multiple iterations. Each iteration uses the fine-tuned model from the previous step as the new baseline, gradually achieving the desired sparsity without significant accuracy degradation. However, one-step pruning uses the EagleEye algorithm [37] to find subnetworks that meet the required pruning ratio and selects the most promising one based on an adaptive batch normalization evaluation. The selected subnetwork is then fine-tuned to recover accuracy without the need for multiple iterations.

In the following figures, we present comparisons of the accuracies achieved by the pruned models versus the base model, as well as the size of the compressed models as a function of the degree of pruning applied. These results allow us to evaluate the trade-off between the reduction in model size and the possible loss of performance introduced by pruning.

Figure 11 shows the accuracy results obtained by the base model and by the different pruned models. When comparing the results, we can clearly see that the models in which the *one-step* method was used obtained outstanding results with respect to the base model, all exceeding the accuracy of the base model, even in the model with 95% of pruning. However, the models in which the *iterative* method was used did not achieve the accuracy of the base model and obtained very poor results compared to the models pruned with *one-step*.

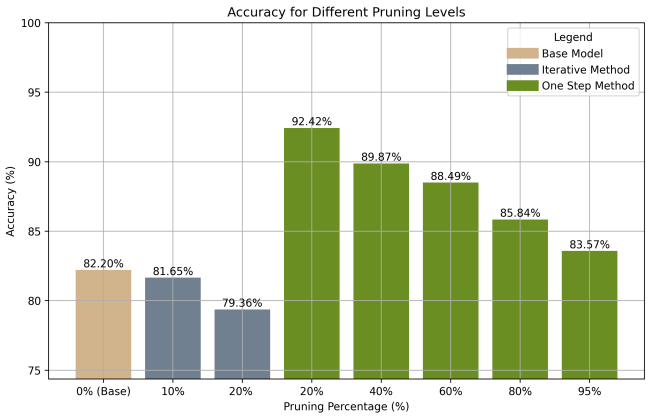

Figure 11: *Accuracy of the original ResNet-50 network and different pruned models with VitisAI.*

Figure 12 shows the sizes of the models in the previous figure. In both *one-step* and *iterative* models, we see that as we increase the degree of pruning, the size becomes smaller and smaller, as expected originally.

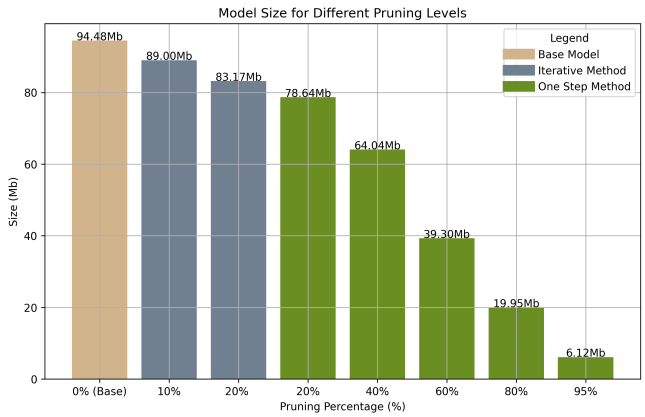

Figure 12: *Size of the original ResNet-50 network and different pruned models with VitisAI.*

After looking at the accuracy and size results of the models, we see that the *one-step* method is much more effective, both in terms of accuracy and size reduction, than the *iterative* method for applying pruning to our Resnet-50 model. This is

caused because the *iterative* method performs iterative pruning followed by rescaling, whereas the *one-step* method performs pruning only once. This form of pruning implemented by the *iterative* method causes the Resnet-50 model, containing batch normalization layers, to suffer a greater loss of accuracy and less size reduction with the *iterative* method than with the *one-step* method.

C. Knowledge Distillation Experimentation

This section presents the results obtained by applying the knowledge distillation technique to transfer learning from a teacher model (ResNet-50) to several lighter student models. This technique allows the creation of compressed models that maintain high accuracy, taking advantage of the information learned by the base model during its training. The experiments performed include distillation to different student model architectures, such as ResNet-18, SqueezeNet, MobileNet, and two models from scratch (with convolutional and fully-connected layers), to compare the effects of this technique. These results are key to understand how knowledge distillation can be used to develop more compact and efficient models, while maintaining adequate performance in tasks such as image classification. The following figures analyze the accuracies achieved by the student models and their final sizes, evaluating the trade-off between performance and efficiency.

Figure 13 shows the accuracy results of the different models. We see that the from scratch models achieve very low accuracy with respect to the base model, whereas the SqueezeNet model manages to match the accuracy of the ResNet-50. On the other hand, the two models ResNet-18 and MobileNet manage to surpass the accuracy of the base model, with the ResNet-18 and MobileNet model standing out above the ResNet18 with intermediate layer distillation, as they achieve a fairly high accuracy with respect to the base model.

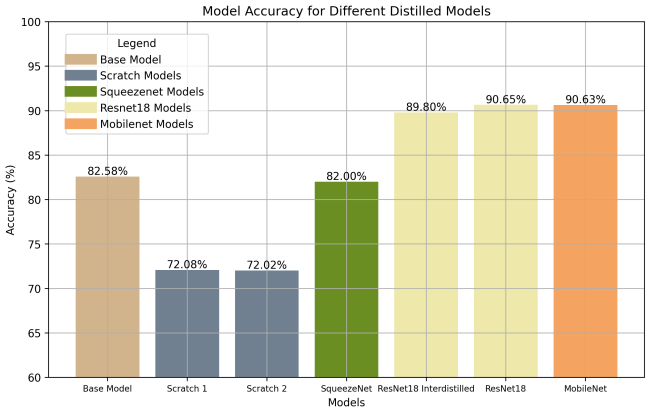

Figure 13: *Accuracy of differents distilled models (Pytorch).*

Figure 14 shows the size of the different distilled models. First, we see that the two ResNet-18 models reduce their size by half with respect to the base model. However, the from scratch models, the SqueezeNet model and the MobileNet model manage to reduce their size significantly with respect to the original model, achieving a reduction of more than 80% of their size, and even more than 95% in the case of the SqueezeNet model.

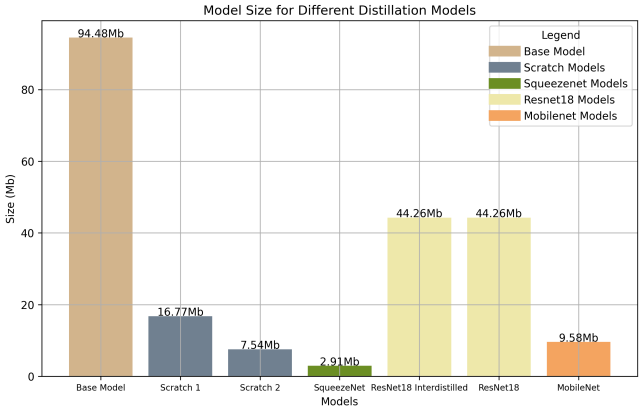

Figure 14: *Size of differents distilled models (Pytorch).*

After analyzing the accuracy and size results of the models, we can analyze more clearly which models have performed well. First, the models from scratch manage to significantly reduce their size, but at the cost of a significant loss of accuracy. However, the other models manage to match or exceed the accuracy of the base model and reduce its size. Within these models we can highlight the SqueezeNet model, which manages to match the accuracy of the base model and is the model that manages to reduce its size the most, which was the desired result of the distillation application. On the other hand, the two ResNet-18 models manage to exceed the accuracy of the base model, but the reduction in size is not as significant as in the other models. Finally, the MobileNet model manages to surpass the accuracy of the base model and obtain a much smaller size than the ResNet-18 models, making it the model with the best balance between accuracy and reduction of the models studied, and although it does not achieve a size as small as the SqueezeNet, it manages to get quite close, but also with the advantage of achieving a higher accuracy.

D. Combination techniques experimentation

In this section we analyze the results obtained by combining quantization, pruning and distillation to compress models. All possible combinations of techniques have been tested, as shown in Figure 4. In this way, we will be able to study whether the combination of different techniques can provide better results, both in terms of accuracy and model size. In addition, all the models presented in this section will be deployed in ZCU104 for further analysis.

The combination of compression techniques has been performed as follows:

- Base Model with One Step pruning of 20% and PTQ.
- Base Model with One Step pruning of 95% and PTQ.
- Distilled SqueezeNet with PTQ.
- Distilled ResNet-18 with PTQ.
- Distilled MobileNet with PTQ.
- Distilled SqueezeNet with Iterative pruning of 20% and PTQ.

- Distilled ResNet-18 with One Step pruning of 20% and PTQ.
- Distilled MobileNet with One Step pruning of 20% and PTQ.

Figure 15 shows the accuracy results of the different models. First, the combination of pruning with quantization on the base model results in both models (20% and 95% pruning) obtaining a higher accuracy than the base model. The increase in accuracy in these models may be due to the fact that the pruning (structured in this case) eliminates the channels that are not necessary, thus reducing the redundancy of the model and making the network focus on the most important aspects of the data. In addition, the fine tuning performed after pruning helps to readjust the remaining weights, thus escaping possible local minima and obtaining better overall results. Furthermore, by including quantization, the reduction in accuracy can also act as a regularization of the model. On the other hand, the fact of applying quantization to the pruned models does not seem to reduce the accuracy obtained by the models in Figure 11. This may be due to the fact that by applying pruning and then fine-tuning, the model has managed to adapt to the lack of information, so it is less sensitive to lose accuracy when quantization is applied.

If we look at the combination of knowledge distillation with quantization, we see that of the three models, only the *ResNet-18 + PTQ* model achieves a higher accuracy than the base model. The remaining two models (*SqueezeNet + PTQ* and *MobileNet + PTQ*) achieve a lower accuracy than the base model. Furthermore, if we look at Figure 13, which shows the accuracy results of only distilled models, we see that the quantized models of SqueezeNet and MobileNet lose accuracy with respect to the same models only distilled. The reason for this loss of accuracy in these two models and not in the ResNet-18 model may be due to the fact that both SqueezeNet and MobileNet are small models and designed to be efficient and compact, so by applying a reduction in accuracy weights with quantization, these models have less margin to compensate for this loss of accuracy.

Finally, pruning has now been applied to the same models mentioned above as an intermediate step between distillation and quantization. It is observed that pruning has significantly improved the accuracy of the models, especially in the SqueezeNet and MobileNet. This improvement in the models may be caused by the fact that by applying pruning the excess connections are reduced, and the fact of applying a fine-tuning to these models means that they can adapt better to the loss of accuracy in the weights, making that especially the SqueezeNet and MobileNet do not lose so much accuracy when they are quantized, as it happened with the distilled and quantized models without pruning.

Figure 16 shows the sizing results of the different models. In the models where pruning and quantization have been applied, if we compare them only with the models with pruning, we see that adding quantization as an additional step in the compression of the models helps to further reduce the final size of the models. The same happens with the models where distillation and quantization were applied, where the resulting models are smaller than the previous models only distilled.

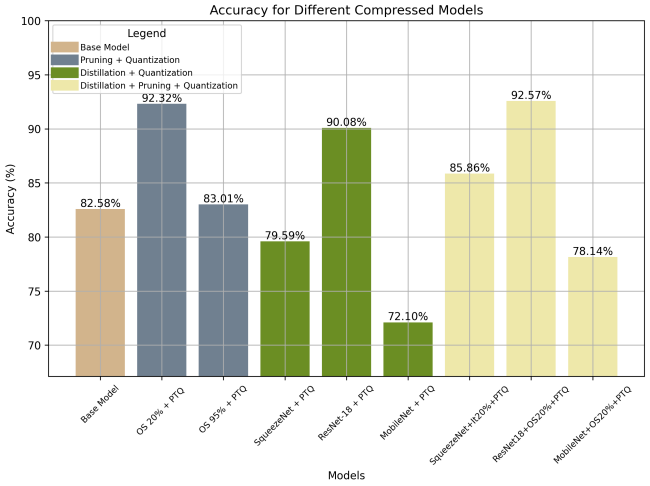

Figure 15: *Accuracy of differents compressed models (Vitis AI).*

Finally, we see that if we apply the three techniques together, the compression power is even greater, obtaining even smaller models, thus fulfilling the results expected a priori.

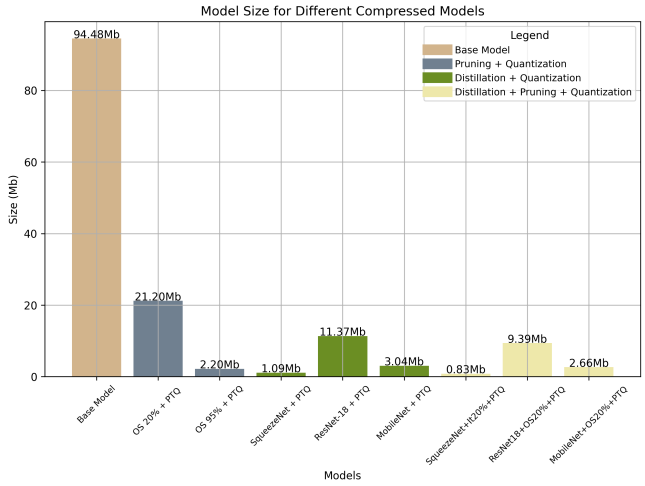

Figure 16: *Size of differents compressed models (Vitis AI).*

After these results, the combination of distillation, pruning, and quantization demonstrates the potential of integrating multiple compression techniques to achieve a balance between model accuracy and size reduction. While each technique individually contributes to different aspects of model optimization, their combination leverages their complementary strengths. Pruning refines the model structure by removing redundancies, distillation transfers knowledge effectively to smaller architectures, and quantization further reduces the computational and memory requirements.

E. ZCU104 Deployment

Finally, in this section we will run several of the models generated with the previous compression processes, and run them on the ZCU104 board, described in Section II-D. The

selected models were: all quantization models and all compressed models showed in Section IV-D. With these models we will run a test with 10,000 images of the CIFAR10 dataset, and we will evaluate the accuracy results obtained, and the processing speed of each model. This will give us a more realistic view of how each model behaves when run on the edge.

Table I shows the results obtained from running the models on the edge. Regarding the quantized models, we see that the best accuracy obtained is the QAT model trained without the weights of the base model, and in terms of processing speed, the three models offer similar results. In the models with Pruning + PTQ, the model with 20% of pruning obtained significantly better accuracy results (10% better) than the model with 95% of pruning. However, the model with 95% of pruning obtained much larger execution speed results (427.55 FPS vs. 91.89 FPS). In the distilled and quantized models we see that the *SqueezeNet + PTQ* model achieves quite similar results to the base model and the *QAT scratch* model in terms of accuracy, however it obtains much better results in terms of execution speed. On the other hand, the *ResNet-18 + PTQ* model obtains much better results than most of the models with 89.56% accuracy and with speed results also quite surprising with 201.09 FPS. On the other hand, the *MobileNet + PTQ* model, although the speed results are quite good, the accuracy results are too poor, with a very large decrease compared to the base model. Finally, in the models with distillation, pruning and quantization, the *SqueezeNet + pruning 20% + PTQ* and *ResNet-18 + pruning 20% + PTQ* models achieve quite good accuracy results, both exceeding the accuracy of the base model, and with quite good execution speeds, both exceeding 200 FPS. However, the *MobileNet + pruning 20% + PTQ* model despite its good execution speed results, its accuracy decreases a little and obtains a lower accuracy than the base model.

TABLE I: *Comparison of different models executed in ZCU104.*

Model	**Accuracy (%)**	**FPS**
PTQ Resnet-50	70.08	83.52
QAT Resnet-50 weights	75.55	83.52
QAT Resnet-50 scratch	78.03	83.50
OS 20% + PTQ	91.84	91.89
OS 95% + PTQ	81.80	427.55
SqueezeNet + PTQ	78.19	323.35
ResNet-18 + PTQ	89.54	201.09
MobileNet + PTQ	66.99	271.61
SqueezeNet + It 20% + PTQ	84.79	352.98
ResNet-18 + OS 20% + PTQ	91.81	215.44
MobileNet + OS 20% + PTQ	75.15	296.96

These results show a trend in the speed of model execution, and that is that the smaller the models, the faster their execution speed. Counterpart, the accuracy of smaller models tends to be lower than larger models. However, with the correct application of the applied compression techniques, we see that it is possible to achieve quite good accuracy results, in many cases exceeding the accuracy of the base model, and with a much higher execution speed.

F. Code and Reproducibility

To facilitate the replication of the experiments performed in this work, the code has been made available in the following repository:

Repository: https://github.com/RomeroRomeroMartin/DeepCompressionOnFPGA.git

V. CONCLUSIONS

This work has demonstrated the effectiveness of various compression techniques applied to deep neural networks for deployment on resource-constrained devices, such as FPGA platforms. The results obtained have shown how quantization, pruning and knowledge distillation can not only significantly reduce the size of the models, but also maintain -and even, in some cases, improve- the accuracy of the predictions.

First, quantization has been confirmed as a key technique to reduce the size of models and optimize their use in constrained environments. Both post-training quantization (PTQ) and quantization-aware training (QAT) have been found to maintain the accuracy of the base model in simple architectures, such as networks with two fully connected layers. In more complex models, such as ResNet-50, QAT-quantized models not only maintained accuracy, but sometimes slightly improved it. This phenomenon can be attributed to the implicit regularization effect that quantization introduces by limiting the accuracy of the weights and activations, which helps to avoid overfitting. However, when Vitis AI was used for quantization, a slight decrease in accuracy was observed compared to models quantized exclusively with PyTorch, suggesting that the Vitis AI implementation could benefit from additional optimizations.

The application of structured pruning stood out as an effective strategy to reduce model complexity by eliminating unnecessary components, such as filters and full channels. In particular, the One-Step method outperformed the iterative approach, achieving more compact models with less impact on accuracy. This approach demonstrated that it is possible to significantly reduce the size of the model while improving its generalizability, highlighting the crucial role of post-pruning fine-tuning.

For its part, knowledge distillation made it possible to transfer learning from a complex model to lighter architectures, such as ResNet-18 and MobileNet. This resulted in models that achieved a remarkable balance between small size and high accuracy. MobileNet, in particular, stood out as an efficient solution, achieving a size reduction of over 80% and accuracy comparable to the base model. However, models trained from scratch, while achieving significant compressions, exhibited notable losses in accuracy, thus highlighting the importance of starting from pre-trained architectures.

When the compression techniques were combined, the results were even more promising. The integration of distillation, pruning and quantization proved to be a powerful strategy for maximizing efficiency without sacrificing accuracy. For example, models such as SqueezeNet and ResNet-18, optimized by combining these techniques, were able to outperform the accuracy of the base model while significantly reducing

its size. This is evidence that these methodologies are not mutually exclusive, but can complement each other to take advantage of their individual strengths.

Finally, the deployment of the optimized models on the ZCU104 FPGA platform validated the feasibility of these techniques in real scenarios. The models not only maintained competitive performance in terms of accuracy, but also delivered high processing rates, reaching execution speeds in excess of 200 frames per second in some cases. These results confirm that it is possible to implement deep neural networks on edge devices without compromising their performance or their ability to perform real-time inference.

VI. Future Work

The results of this work open multiple lines of research to explore new ways to optimize and deploy deep learning models on edge devices. One area of immediate interest is the improvement of the quantization process in Vitis AI, since, although the models quantized with this platform were functional, their accuracy fell below that obtained with PyTorch. This raises the need to investigate alternative workflows or more advanced settings to minimize these performance losses.

Another promising direction is the extension of the use of compression techniques to more complex architectures and more demanding tasks. For example, applying these methodologies to object detection and semantic segmentation networks would allow evaluating their effectiveness in practical scenarios where size reduction and inference time are critical. In addition, these networks tend to be larger and more computationally demanding, which makes the implementation of techniques such as quantization and pruning even more relevant.

Finally, it would be valuable to explore how these optimizations impact not only model size and accuracy but also power consumption. This aspect is crucial for edge devices operating under limited autonomy conditions, such as IoT sensors and embedded systems. Recently, the success of DeepSeek [38] has demonstrated the potential of model size reduction as a key factor in improving efficiency without sacrificing performance. Unlike its competitors, DeepSeek achieves state-of-the-art results while maintaining a significantly smaller architecture, highlighting the growing importance of compact and energy-efficient AI models. As large-scale data centers become increasingly unsustainable due to their enormous energy demands, the need for Green AI [39] solutions has never been more pressing.

References

[1] N. C. Thompson, K. H. Greenewald, K. Lee, and G. F. Manso, "The computational limits of deep learning," *CoRR*, vol. abs/2007.05558, 2020. [Online]. Available: https://arxiv.org/abs/2007.05558

[2] Y. Mao, X. Yu, K. Huang, Y.-J. A. Zhang, and J. Zhang, "Green edge ai: A contemporary survey," 2024. [Online]. Available: https://arxiv.org/abs/2312.00333

[3] F. Samie, V. Tsoutsouras, D. Masouros, L. Bauer, D. Soudris, and J. Henkel, "Fast operation mode selection for highly efficient iot edge devices," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 3, pp. 572–584, 2020.

[4] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, "A white paper on neural network quantization," *CoRR*, vol. abs/2106.08295, 2021. [Online]. Available: https://arxiv.org/abs/2106.08295

[5] D. W. Blalock, J. J. G. Ortiz, J. Frankle, and J. V. Guttag, "What is the state of neural network pruning?" *CoRR*, vol. abs/2003.03033, 2020. [Online]. Available: https://arxiv.org/abs/2003.03033

[6] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015. [Online]. Available: https://arxiv.org/abs/1503.02531

[7] J. Ren, G. Yu, Y. Cai, and Y. He, "Latency optimization for resource allocation in mobile-edge computation offloading," *IEEE Transactions on Wireless Communications*, vol. 17, no. 8, pp. 5506–5519, 2018.

[8] J. Zhang and R. Saab, "Spfq: A stochastic algorithm and its error analysis for neural network quantization," 2023. [Online]. Available: https://arxiv.org/abs/2309.10975

[9] 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. 1–20, 2024.

[10] L. Hedegaard, A. Alok, J. Jose, and A. Iosifidis, "Structured pruning adapters," *Pattern Recognition*, vol. 156, p. 110724, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320324004758

[11] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, "Fitnets: Hints for thin deep nets," 2015. [Online]. Available: https://arxiv.org/abs/1412.6550

[12] L. Zhang, C. Bao, and K. Ma, "Self-distillation: Towards efficient and compact neural networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 8, pp. 4388–4403, 2022.

[13] Z. Yang, A. Zeng, Z. Li, T. Zhang, C. Yuan, and Y. Li, "From knowledge distillation to self-knowledge distillation: A unified approach with normalized loss and customized soft labels," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 17 185–17 194.

[14] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[15] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[16] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, 05 2012.

[17] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database."

[18] S. Brown and J. Rose, "Field-programmable gate arrays," *IEEE Design & Test of Computers*, 1996.

[19] A. Nechi, L. Groth, S. Mulhem, F. Merchant, R. Buchty, and M. Berekovic, "Fpga-based deep learning inference accelerators: Where are we standing?" *ACM Trans. Reconfigurable Technol. Syst.*, 2023. [Online]. Available: https://doi.org/10.1145/3613963

[20] Xilinx, Inc., *ZCU104 Evaluation Board User Guide*, oct 2018, accessed: 2024-11-26. [Online]. Available: https://www.mouser.com/datasheet/2/903/ug1267-zcu104-eval-bd-1596428.pdf

[21] AMD Xilinx, Inc., *Vitis AI User Documentation*, 2024, version 3.5, Accessed: 2024-11-26. [Online]. Available: https://xilinx.github.io/Vitis-AI/3.5/html/index.html

[22] Y. Fukuda, K. Yoshida, and T. Fujino, "Evaluation of model quantization method on vitis-ai for mitigating adversarial examples," *IEEE Access*, vol. 11, 2023.

[23] R. Yarnell, M. Hossain, and R. F. DeMara, "Image quantization tradeoffs in a yolo-based fpga accelerator framework," in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, 2023, pp. 1–7.

[24] M. Nobari and H. Jahanirad, "Fpga-based implementation of deep neural network using stochastic computing," *Applied Soft Computing*, vol. 137, p. 110166, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494623001849

[25] G. Tatar, S. Bayar, and . iek, "Real-time multi-learning deep neural network on an mpsoc-fpga for intelligent vehicles: Harnessing hardware acceleration with pipeline," *IEEE Transactions on Intelligent Vehicles*, vol. 9, no. 6, pp. 5021–5032, 2024.

[26] P. Team. (2024) Pytorch: An open source machine learning framework. Accessed: 2024-12-01. [Online]. Available: https://pytorch.org/

[27] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[28] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:18268744
[29] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385
[30] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size," 2016. [Online]. Available: https://arxiv.org/abs/1602.07360
[31] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," 2017. [Online]. Available: https://arxiv.org/abs/1704.04861
[32] AMD, "Dpuczdx8g for zynq ultrascale+ mpsocs product guide (pg338)," 2025, accesed: January 24, 2025. [Online]. Available: https://docs.amd.com/r/en-US/pg338-dpu/Introduction?tocId=3xsG16y_QFTWvAJKHbisEw
[33] M. Nagel, M. van Baalen, T. Blankevoort, and M. Welling, "Data-free quantization through weight equalization and bias correction," 2019. [Online]. Available: https://arxiv.org/abs/1906.04721
[34] I. Hubara, Y. Nahshan, Y. Hanani, R. Banner, and D. Soudry, "Improving post training neural quantization: Layer-wise calibration and integer programming," 2020. [Online]. Available: https://arxiv.org/abs/2006.10518
[35] AMD, "Iterative pruning - vitis ai user guide," 2025, accesed: January 7, 2025. [Online]. Available: https://docs.amd.com/r/en-US/ug1414-vitis-ai/Iterative-Pruning
[36] ——, "One-step pruning - vitis ai user guide," 2025, accesed: January 7, 2025. [Online]. Available: https://docs.amd.com/r/en-US/ug1414-vitis-ai/One-Step-Pruning
[37] B. Li, B. Wu, J. Su, G. Wang, and L. Lin, "Eagleeye: Fast sub-net evaluation for efficient neural network pruning," 2020. [Online]. Available: https://arxiv.org/abs/2007.02491
[38] D.-A. et al., "Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model," 2024. [Online]. Available: https://arxiv.org/abs/2405.04434
[39] Y. I. Alzoubi and A. Mishra, "Green artificial intelligence initiatives: Potentials and challenges," *Journal of Cleaner Production*, vol. 468, p. 143090, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0959652624025393