CSE145/237D - Final Report on Compressing ResNets for FPGAs

Doheon Lee, Gabriel Marcano

I. Introduction

The contents of this report are intended to be inserted or adapted for the SkipTrim paper.

We have implemented the Skipper algorithm as described by the aforementioned paper in Keras, and have collected data demonstrating that the Keras implementation is no better than regular training for smaller networks. We have also collected validation accuracy data using knowledge distillation as an additional point of comparison, and identified that is no discernible difference for smaller networks between knowledge distillation and Skipper. However, Skipper performs better than knowledge distillation for ResNet56.

II. Keras Implementation

The hls4ml framework supports converting Keras models to FPGAs bitstreams, so we implemented our Skipper algorithm in Keras in order to facilitate this conversion process. However, implementing Skipper in Keras was non-trivial due to limitations in the framework. The original Skipper algorithm relies on modifying the ResNet during the training process, but Keras does not allow for models to be modified once they are instantiated. We worked around this limitation by running a full knowledge distillation training run per Skipper iteration, transferring the weights of the previous student to the next one in the process. Each modification requires us to re-instantiate a brand new model, and to reload weights. One downside of this approach is that we lose the optimizer state every iteration, as Keras does not allow transferring the optimizer state from one model to another. We suspect that it is due to this loss in optimizer state that we were required to let every Skipper iteration run for a complete knowledge distillation training cycle. Figure 1 shows the pseudocode for the Keras implementation.

Fig. 1. Keras Skipper implementation pseudocode

```
skips_per_iteration = skips to remove per iteration    ▷ Must be a multiple of 3 or 1
total_skips = (x - 2) / 2    ▷ x is the number of layers in the ResNet
skips = 0
while skips ≤ total_skips do
    clear Keras memory and models
    teacher = pre-trained resnetx
    student = resnetx with "skips" number of skip connections removed
    if skips == skips_per_iteration then
        student_weights = teacher_weights
    else
        student_weights = prev_student_weights
    end if
    student.load_weights(student_weights)
    distiller = Distiller(teacher, student)
    distiller.fit()    ▷ Go through knowledge distillation
    prev_student_weights = distiller.student.weights
    skips += skips_per_iteration
end while
```

Our Keras Skipper implementation removes skips connections from the top of the model first, progressing downwards until all skip connections are removed.

Keras has an additional limitation that complicated the implementation of Skipper, namely that all layers across all models in memory must be globally unique. The problem with this requirement is that Keras does not allow a good way to transfer weights from models of different shapes, like from one student to the next, unless the students' layers agree on the naming of each layer. We decided to bypass the problem by enforcing our own naming conventions on all of our models and layers. Our naming convention consists of the model name, three underscore characters, followed by the layer name (hls4ml expects layer names to be valid C++ type names, hence the use of multiple underscores to act as a separator). Figure 2 shows an example of the naming convention in use. This way, each model can have layers with the same name, but globally they are distinct, thanks to the model name prefix, from the point of view of Keras.

Keras has routines for loading weights by name, but these require for the names of the layers in the saved weights to match exactly the names of the layers of the model being loaded into. If the layers being loaded to do not match the names of the weights being loaded, there is no feedback from Keras that weights could not be loaded. We implemented our own weight loading routines to have more control over naming mismatches, and also to conform to our new layer naming convention. These routines allows us, as an example, to transfer the weights from the layers on the left part of the top model in figure 2 to the layers of the bottom model, since the layer names match per our naming convention.

So far, we have only tested the Keras implementation with the CIFAR10 dataset.

Our ResNet implementation conforms to the original ResNet architecture as described in [1]. It is possible to combine adjacent batch-normalization and convolutional layers after skips have been removed, but we did not apply this optimization.

We implemented three different training processes in total. We implemented typical model training, to train our teacher

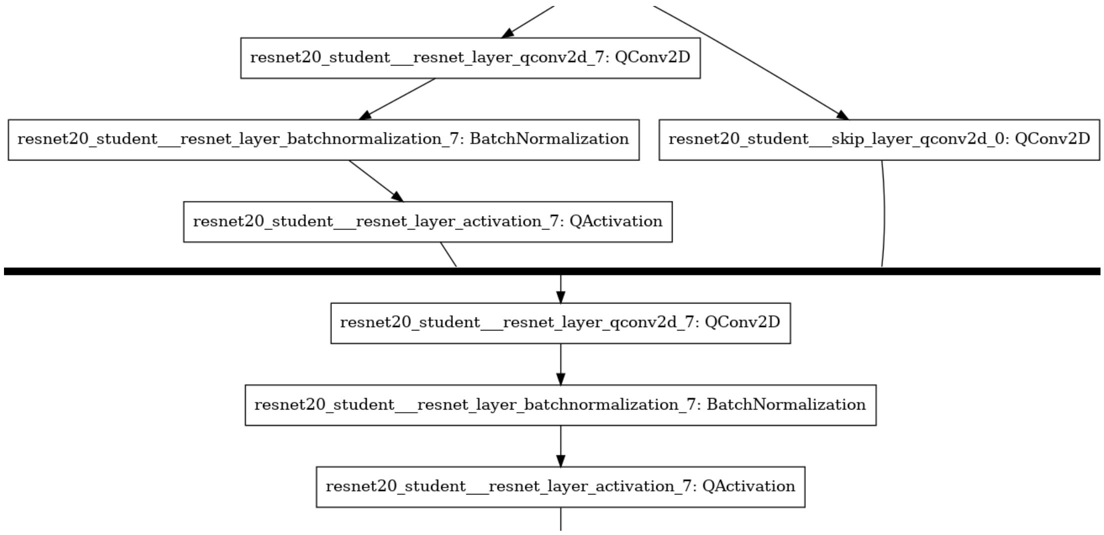


Fig. 2. Sampling of layers from a teacher and a student ResNet20, showing the naming convention implemented to deal with Keras requirements. The top graph belongs to a ResNet20 model with no skips removed, and the bottom belongs to one with some skips removed. In particular, the bottom one is missing the skip_layer_qconv2d_0 layer present in the top one.

ResNets, training using knowledge distillation to use as comparison, and Skipper. Our typical training implementation is configured to run for 200 epochs, with the learning rate reduced periodically. For both our knowledge distillation process and Skipper implementations we leveraged Keras callbacks to help track the rate of progress and make adjustments as needed. Specifically, we configured the learning rate to adjust automatically after 10 epochs with no improvement to the validation accuracy, and to terminate before reaching 200 epochs if there is no improvement in the validation accuracy after 39 epochs. All other parameters for the Skipper implementation loss function are as described in the Skipper section.

Finally, we leveraged hls4ml to convert some models to FPGA bitstream. All of our models, regardless of their quantization parameters during training, were converted to FPGA bitstream using $<16, 8>$ quantization.

III. Results and Discussion

We trained non-quantized and quantized models to compare against the base PyTorch performance. We then implemented the Skipper algorithm to compare against PyTorch. Additionally, we collected some data showing the performance of using knowledge distillation to train a student network by simply removing all skip connections.

A. Training Non-quantized Models

We began our experiments by collecting data for non-quantized models to check that our implementation was working as expected. We did not run Skipper for these experiments, but instead compared the validation accuracy of the teacher model versus that of a student model with all skip connections removed trained using knowledge distillation, and of a model also with all skip connections removed trained normally. Per table I, the teacher model outperformed both other training approaches for all three models tested. For ResNet20 and ResNet32, training the model without skip connections normally yielded better results than via knowledge distillation. ResNet110, on the other hand, knowledge distillation yielded distinctly better results than training a model without skip connections.

TABLE I

Training results for non-quantized models. Teachers were trained normally and did not have any skip connections removed. The students for knowledge distillation had all of their skip connections removed, then trained by the teacher. The skip-less test took the ResNet model, stripped it of all of its skip connections, and then proceeded to train it normally.

	Accuracy (%)		
Model	Teacher	Knowledge Distillation	Skip-less
ResNet20	90.68 ± 0.12	88.97 ± 0.19	90.01 ± 0.19
ResNet32	91.61 ± 0.17	88.33 ± 0.19	89.59 ± 0.30
ResNet110	92.57 ± 0.14	79.14 ± 1.78	48.81 ± 20.32

For ResNet20 and ResNet32, there are differences between each different training processes, but for ResNet20 it is no more than around 0.6%, and for ResNet32 it is no more than around 3%. This suggests that out of these two compression approaches, training these smaller ResNets normally, after removing all of their skip connections, is the better approach for compressing the ResNets for the CIFAR10 dataset.

For ResNet110, the the removal of skip connections incurs a noticeable decrease in validation accuracy. This is expected, as the purpose of skip connections is to aid convergence for deeper networks. What is noteworthy from this data is the difference between knowledge distillation and training the skip-less variant of ResNet110. The knowledge distillation approach yielded a drop in accuracy of around 13%, with a relatively small standard deviation. However, the regular training approach for the skip-less ResNet110 yielded a validation accuracy drop of around 43% with a large standard deviation of 20%. This indicates that either 200 epochs were not enough to allow the network to converge, or that without skip connections the network was not able to converge properly on optimal solutions.

B. Training Quantized Models

The limitations of Keras and Tensorflow that we encountered as we implemented Skipper dramatically increased the amount of time required to run it, leading to an increase between five to nine times that of regular training. This increase was initially unexpected, and reduced the amount of data we could collect in a reasonable timeframe.

TABLE II
TRAINING RESULTS FOR QUANTIZED MODELS. TEACHERS WERE TRAINED NORMALLY AND DID NOT HAVE ANY SKIP CONNECTIONS REMOVED. THE STUDENTS FOR BOTH KNOWLEDGE DISTILLATION AND SKIPPER HAD ALL OF THEIR SKIP CONNECTIONS REMOVED BEFORE STARTING THE RESPECTIVE ALGORITHM.

	Accuracy (%)		
Model	Teacher	Knowledge Distillation	Skipper
ResNet20 <8,3>	91.56 ± 0.15	90.22	91.32
ResNet20 <16,4>	91.79	90.19	90.93
ResNet56 <8,3>	92.14	82.11	86.78

We trained some quantized models with knowledge distillation and with Skipper in order to compare their performance. As shown in table II, regular knowledge distillation fared better in for ResNet20, but Skipper yielded better validation accuracy for ResNet56. Of note, we removed one skip connection at a time for ResNet20, but we removed 3 skip connections at a time for ResNet56, in order to reduce the execution time of the test.

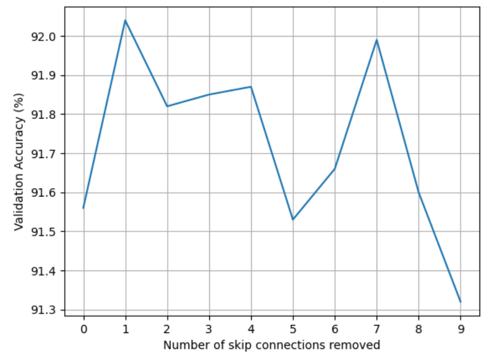
The graphs in figure 3 show the progression of Skipper on the three models tested with Skipper. Notably, all three models show a downward trend in accuracy. ResNet20 <8,3> shows a slightly erratic trend, but looking at the scale, the graph encompasses a very small range. Resnet <16,4> and ResNet56 <8,3> show more notable decreases, although interestingly both show a small relative increases in validation accuracy towards the end of the algorithm.

C. FPGA Results

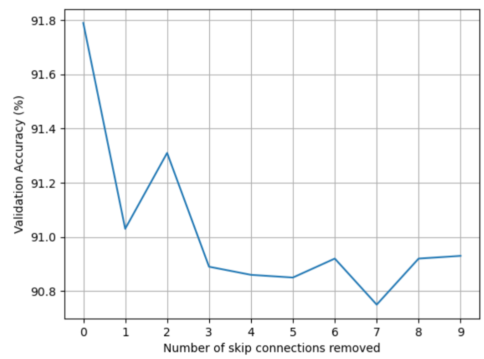
TABLE III
FPGA CONVERSION RESULTS, SHOWING THE PERCENT UTILIZATION OF RESOURCES OF THE ALVEO U250 FPGA.

Design, Quantization	Val. Acc. (%)	Utilization (%)			
		BRAM	DSP	FF	LUT
ResNet20 (full)	90.68 ± 0.12	127	12	21	93
ResNet20 <8,2> (Skipper student)	91.76	104	14	20	98

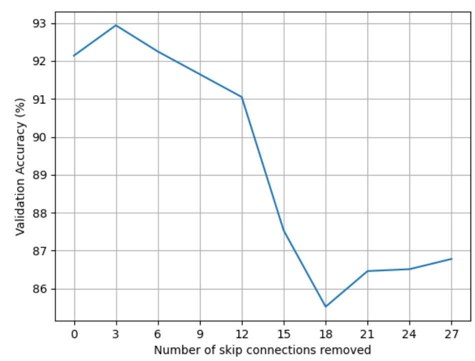
Table III summarizes our findings. Our initial work shows that so far, even removing all skip connections only ResNet20 models are small enough to fit our Alveo U250 FPGA target. One major caveat is that we did not adjust the precision parameter in our configuration files from <16,8> to values corresponding to the quantization used during training.



(a) ResNet20 <8,3> Skipper



(b) ResNet20 <16,4> Skipper



(c) ResNet56 <8,3> Skipper

Fig. 3. Validation accuracy of multiple ResNets while trained by the Skipper algorithm.

IV. FUTURE WORK

Our Keras implementation experiments were all performed using the CIFAR10 dataset. We need to run similar tests with other datasets to ensure that our results are not coupled to CIFAR10 specifically.

While we have validation accuracy reports for the Keras models, we have not validated that the converted models perform similarly once instantiated on an FPGA. We need to simulate the converted bitstreams, and for the ones that are able to fit inside physical FPGAs, instantiate them and perform these experiments on hardware.

One of the more promising approaches for reducing resources utilization for these neural network models on FPGAs is by quantizing weights. Our experiments cover some common quantization cases, but we need to explore to see if smaller quantization values are viable, and what their impact on the model accuracy is. Additional work is also needed to determine if it is possible to further optimize quantization by assigning different quantization parameters to different parts of the network (e.g. having different parameters for weights and for activation computations).

Our initial tests using knowledge distillation indicates that it is better than basic training for deeper networks. We need to collect more data to compare knowledge distillation with Skipper. Additionally, we need to implement Trimmer in Keras and compare its performance against Skipper and against knowledge distillation. We expect Trimmer to require similar adjustments as with Skipper due to Keras limitations.

One aspect of our data collection that was not well controlled was our batch sizes for training. We need to explore the impact on validation accuracy of changing and mixing batch sizes for training the teacher and training the student with Skipper and/or knowledge distillation.

We begun exploring the conversion process from Keras models to FPGA bitstreams, but there are still many parameters to consider. There may be better ways to quantize models to improve FPGA utilization, and there may be ways to further optimize models prior to conversion (such as by combining batch-nomarlization and convolution layers once skip connections are removed).

We need to identify what is different between our Keras and PyTorch implementations to identify if the difference in Skipper behavior is due to the optimizer state not being retained between iterations in Keras, or other Keras specific details, or if it is due to a deeper, underlying misunderstanding of the mechanisms at play. One way to confirm that the problem is with the Keras implementation would be by re-implementing Skipper in yet another framework, but one that allows modifying networks while they are being trained. A second alternative would be to implement Skipper in PyTorch in a similar manner as we have done for Keras (doing full knowledge distillation training for each skip connection removed), and confirm that we see similar results as with Keras.

V. Conclusion

We implemented Skipper and knowledge distillation with Keras, and observed that Skipper does not perform as well as with our PyTorch implementation. For smaller networks Skipper nor knowledge distillation yield better results than training a model with no skip connections, but as the networks grow deeper, Skipper beings to perform better than knowledge distillation, and for larger models knowledge distillation performs better than regular training with no skip connections.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.