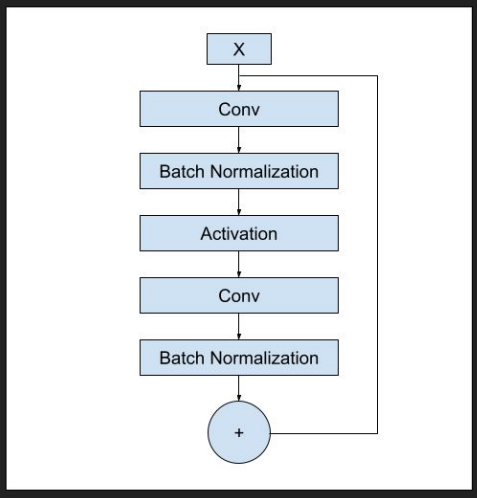
Compressing ResNets for FPGAs

Doheon Lee, Gabriel Marcano

Introduction



Example of a Resnet layer

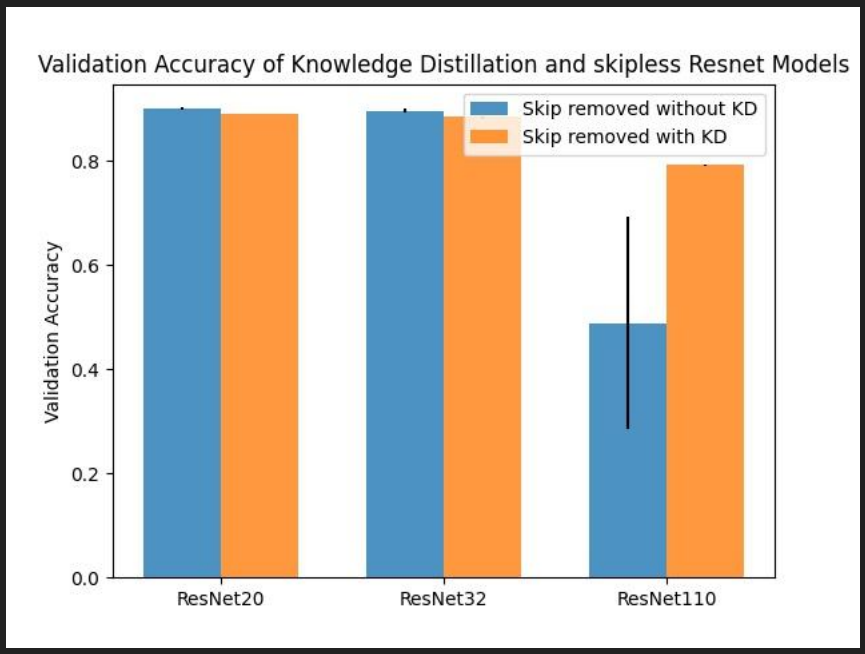
Example of a FPGA Hardware

- Neural networks are used in a lot of applications, but require power-hungry GPUs. FPGAs are an alternative, but they have limited sizes. We need to compress neural networks somehow to get them to fit.
- One approach is to Knowledge distillation: have a teacher network teach the student how to act like it. Student can be a much smaller network, hence the knowledge is distilled down to the student.
- For this project, we are working with ResNets. There is a promising algorithm that iteratively modifies the student network by removing ResNet skip connections while it is being trained and is supposed to yield better results than regular knowledge distillation-- we implemented a version of this algorithm for some of our tests.

Results: Non-quantized

Trained model	Mean (Top 1)	Standard deviation (Top 1)
ResNet20	0.9068	0.0012
ResNet32	0.9161	0.0017
ResNet56	0.9241	0.0024
ResNet110	0.9257	0.0014

These are the non-quantized ResNets we tested earlier on. As expected, the increasing number of ResNet layers results in a higher accuracy.



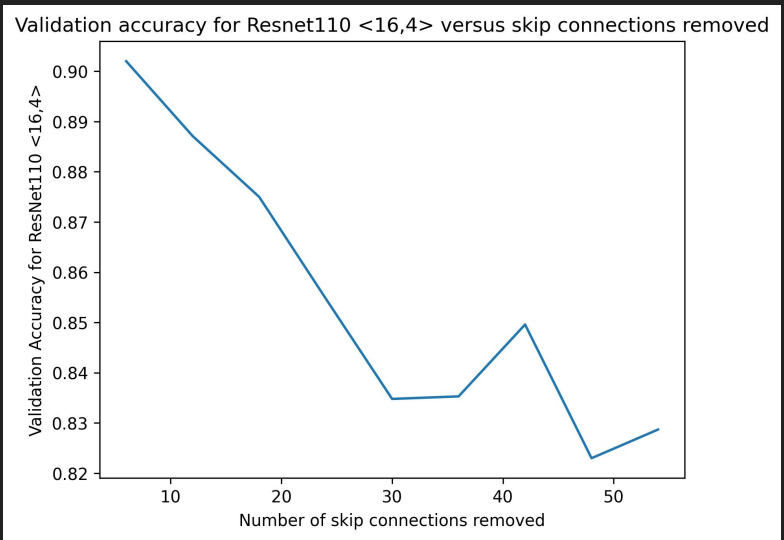
This is the result of training of the skip-less versions with no KD and skip-less versions with KD. The trend is going lower in accuracy as ResNet layers increase. For the smaller ones, ResNet20 and 32, training was better than KD, but for ResNet110, training was absolutely worse. This makes sense because the point of these skip connections, per the original ResNet paper, is to help networks converge on a solution.

Results: Quantized

Trained model	Mean (Top 1)	Standard deviation (Top 1)
ResNet20 <8, 3>	0.9156	0.0015
ResNet56 <8, 3>	0.9218 (1 sample)	N/A
ResNet56 <16, 4>	0.9242	0.0016
ResNet110 <8, 3>	0.9275 (1 sample)	N/A
ResNet110 <16, 4>	0.9264	0.0013

In a different set of results, here we quantized some models and trained them normally. Their accuracy follow the general expected trend of the deeper models which performs better with an increase in number of layers, although there isn't a large difference between them. Interestingly enough, these models perform better than the non-quantized variants.

Results: Quantized, iterative



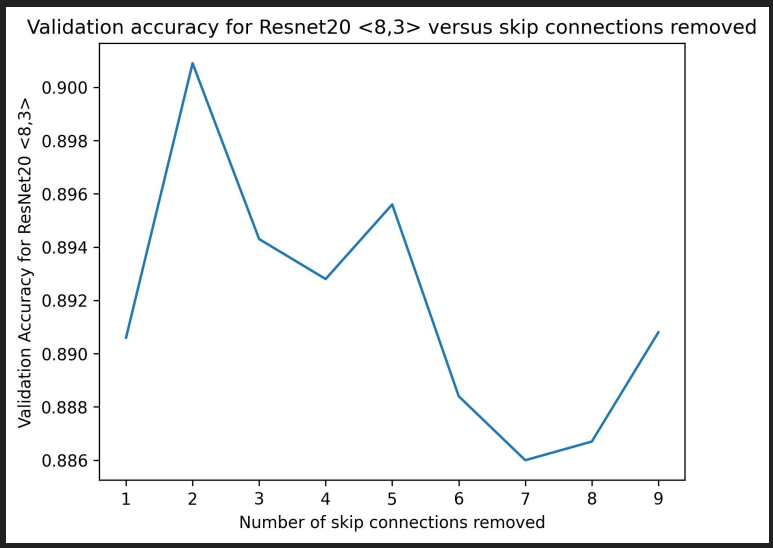
Now, these next two graphs are the results of our implementation of the iterative skip removal algorithm.

This is the graph for the validation accuracy of ResNet110 <16 bits for weights, with 4 bits for the integer portion of the weights> versus the number of skip connections removed.

For reference the teacher (all skip connections intact) has a validation accuracy (Top 1) of 92.46%.

In prior work, accuracy only decreased by around 3.5% compared to the teacher. Here we have more than double that.

Results: Quantized, iterative (cont.)



This is the graph for the validation accuracy of ResNet20 <8 bits for weights, with 3 bits for the integer portion of the weights> versus the number of skip connections removed.

`For reference, the teacher has an validation accuracy of 91.55%.` The trend here is not as clear, but it is still downward.

Here we have a decrease in accuracy of around 2% compared to the teacher, which is not what prior work showed (1% increase).

FPGA quantization results

- So far, can’t find any that fits our FPGA
- Closest is ResNet20 <8, 3>:

Name	BRAM_18K	DSP48E	FF	LUT	URAM
Utilization (%)	104	15	20	98	0

We still have a lot of data to collect, but so far, we haven’t found any design that fits our FPGA. The closest is what you see on the slide, with Block RAM requiring 104% of maximum block ram availability.

We need to review what is going on under the hood, as talking with others that have done this ResNet20 <8, 3> should have fit in this FPGA.

What could possibly go wrong?

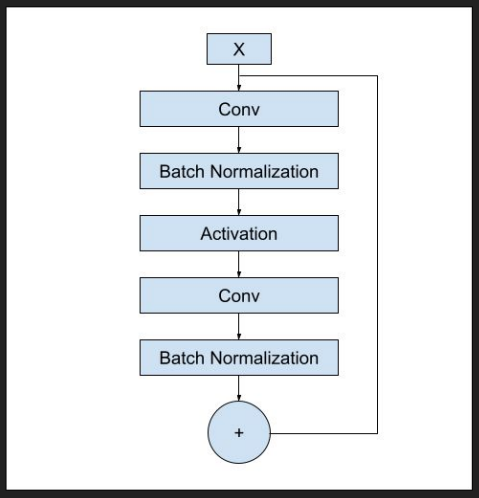
- Skip-removal is 5-9x slower than expected
- Keras is great, until it isn’t
  - Can’t modify model during training
  - Silent weight loading failure (?!?!)

That said, due to limitations in the framework we used, Keras, the iterative approach took between 5 and 9x longer to run (as we were forced to run a full KD training cycle per iteration).

Keras is a relatively nice framework all in all, but but the reason we had to adjust our algorithm was that the algorithm required modifying the network while it is being trained, which Keras/Tensorflow (the backend) doesn’t support.

We just recently found a nasty bug where Keras was silently failing to load weights for the iterative approach. This means that our current data might just be showing the results of running regular KD for each skip removed.

Conclusions



- Non-quantized, KD works better than training for large models
- Iterative skip removal results for large models sort of match prior work, doesn’t match for smaller networks
- Data collection takes much, much, much longer than anticipated
- Beware silent failures!

Questions?