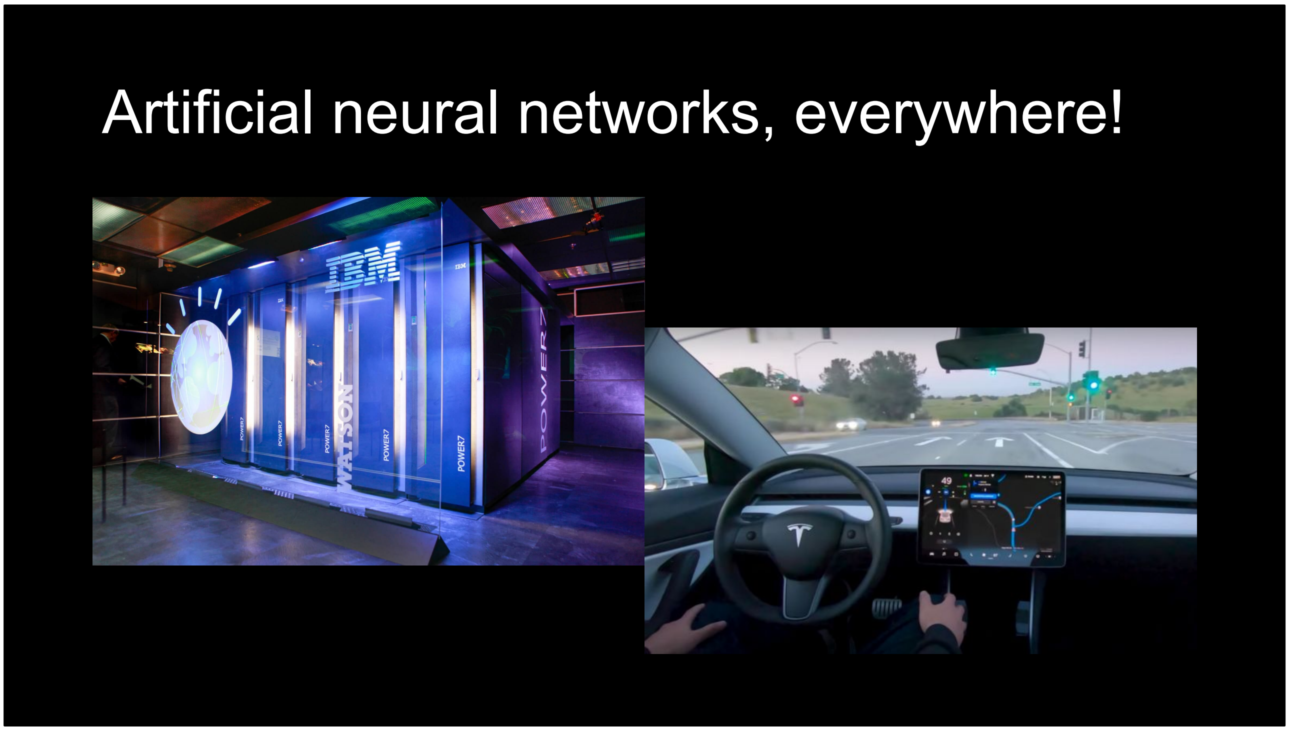
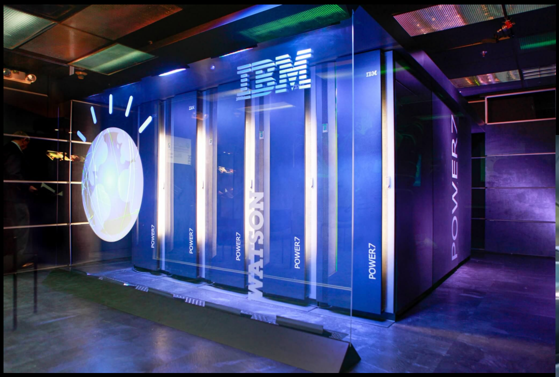


Artificial neural networks, everywhere!



(Ideal)
This is the FPGA project
Just a brief introduction to artificial neural networks, ANNs are digital constructs inspired by our own brains that layer neuron-like components on top of each other and are then trained to perform specific tasks, such as object recognition and identification. We use neural networks, today, to help autonomous cars and trucks drive, to monitor our health and alert us to problems (such as falling asleep at the wheel), to handle voice recognition for home automation systems, and even help diagnose cancer.

Sounds great, right?

So, what’s the catch?

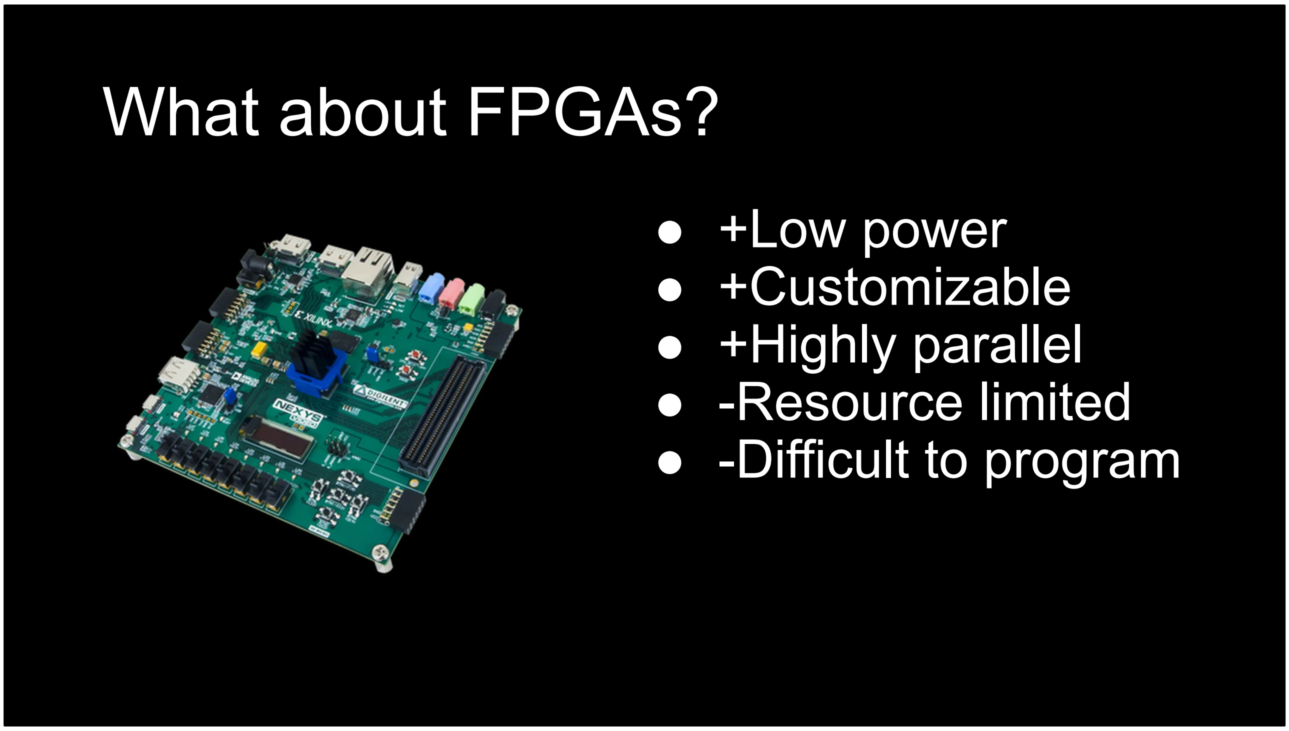
(reality + problem)

So, what’s the catch? If they’re so useful, why aren’t they ubiquitous? Well, training and running current ANNs requires large amounts of computations, which require single or multiple Graphics Processing Unit (GPUs). GPUs are notoriously power hungry, limiting their use to PC’s, datacenters, and other larger devices and additionally they’re HUGE!. What we would like to do is be able to run NNs on small sensors (also called edge devices) out in the field or have devices that you can easily carry around. If we can’t use GPUs, what are our alternatives?

--

So, what’s the catch? If they’re so useful, why aren’t they ubiquitous? Well, training and running current ANNs requires large amounts of computations, and are traditionally run on Graphics Processing Unit (GPUs) that can do these operations on a reasonable timeframe. Alright, so why not deploy GPUs everywhere, then? GPUs are notoriously power hungry, limiting their use to PC’s, datacenters, and other larger devices. As you can see on the slide, they’re big, and even if you could power them with a large battery, you might not want to be carrying that (and the battery) on your back on an trip or an scientific expedition. What we would like to do is be able to install small sensors (also called edge devices) out in the field or have devices that you can easily carry around, running neural networks. If we can’t use GPUs, what are our alternatives?

Picture from https://twitter.com/hardwarecanucks/status/717784884689776640



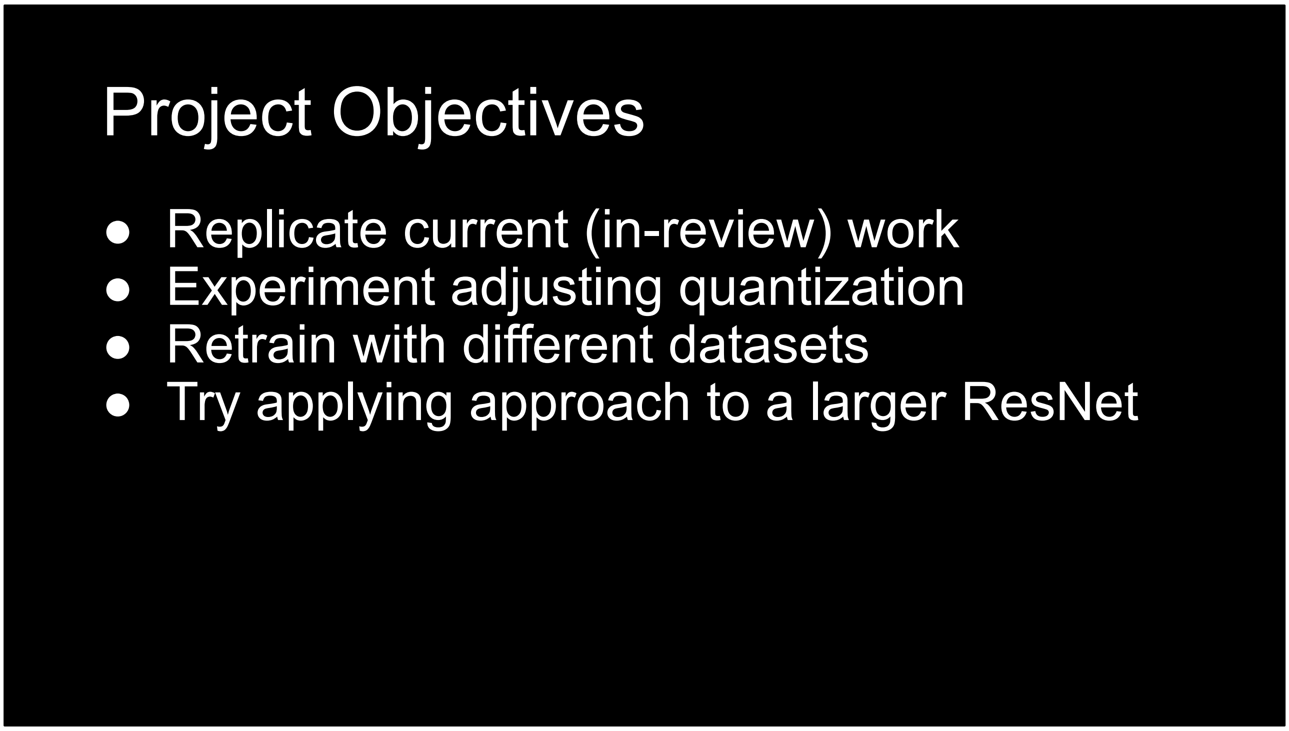
What about FPGAs?

- +Low power
- +Customizable
- +Highly parallel
- -Resource limited
- -Difficult to program

(Solution)

Well, what about FPGAs? An FPGA, or Field Programmable Gate-Arrays, are a collection of digital logic circuits that can be rewired or reprogrammed over and over and over again. FPGAs are low power, customizable, and highly parallel (not like traditional CPUs). That sounds great! But, the reality is that FPGAs are more difficult to program than traditional CPUs and GPUs and have a limited resources that can be used at once. To make the matter worse for FPGAs, neural networks generally use floating point numbers for their weights (think of the knowledge learned by the network), which are expensive to represent on FPGAs.

So, what can we do?



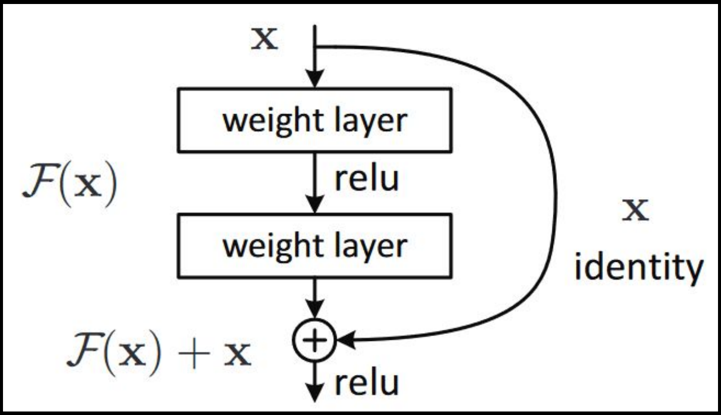
Project Objectives

- Replicate current (in-review) work
- Experiment adjusting quantization
- Retrain with different datasets
- Try applying approach to a larger ResNet

Our proposed solution builds on existing UCSD research, which took a ResNet20 (a NN with skip connections, or connections between non-adjacent layers) and optimized it for FPGA use by quantizing the weights (what this means is representing them as fixed point numbers instead of floating point)

Replicate (in-review) work

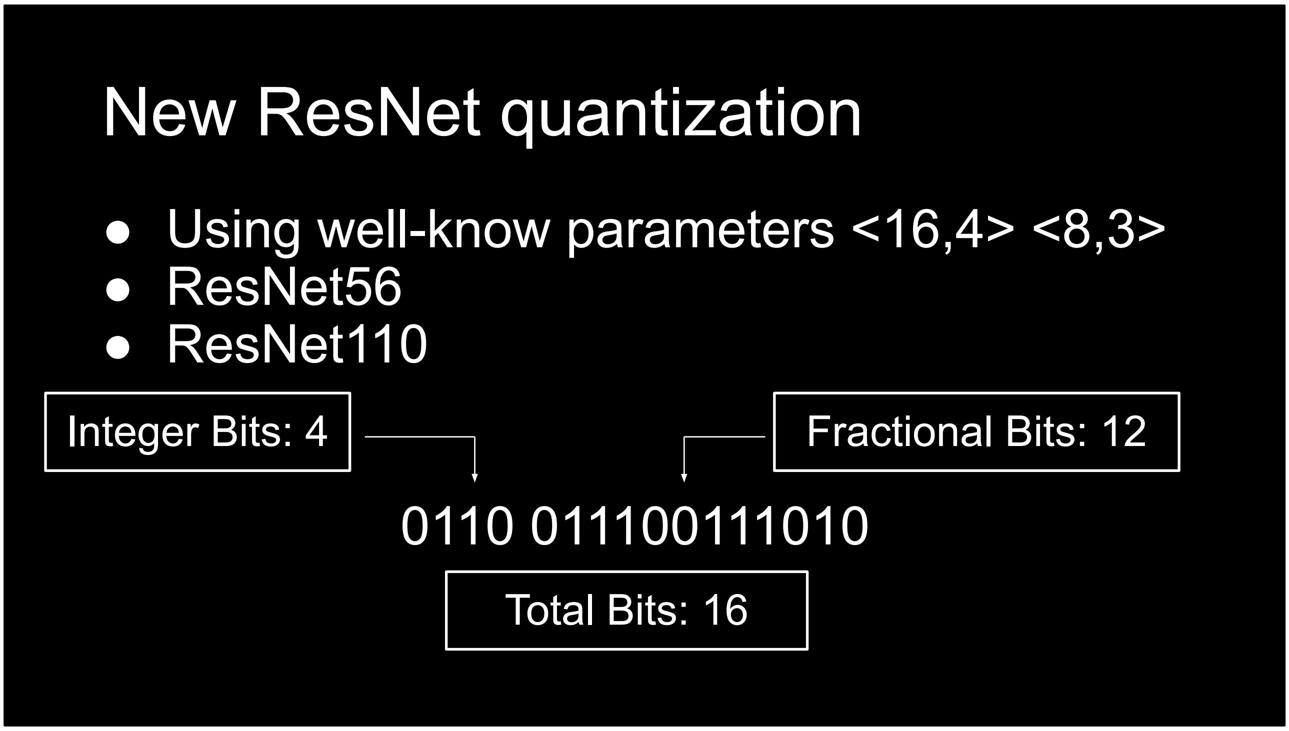
- Skip!
- ResNet20



The current work does two things that we are trying to replicate: implement Skip, and train ResNet20 with skip and converting the result to an FPGA bitstream.

The picture on the slide is an pictorial representation of what skip connections are in a network. Generally, data from one layer flows to the next layer, but with skip connections some data is allowed to skip and recombine with the output of later layers. Skip connections can help the network converge into a solution.

The Skip algorithm is interesting, in that it attempts to simplify the network by removing skip connections and leveraging knowledge distillation. Knowledge distillation is a method of compressing neural networks that relies on a teacher network. We can think of it like teaching a student network how to be more like the teacher. Specifically, Skip takes a regular ResNet as a teacher, and takes another ResNet and cuts off some skip connections and makes this the student. We will be exploring what effect this has on FPGA resource utilization and on the accuracy of the resulting skipped ResNet.



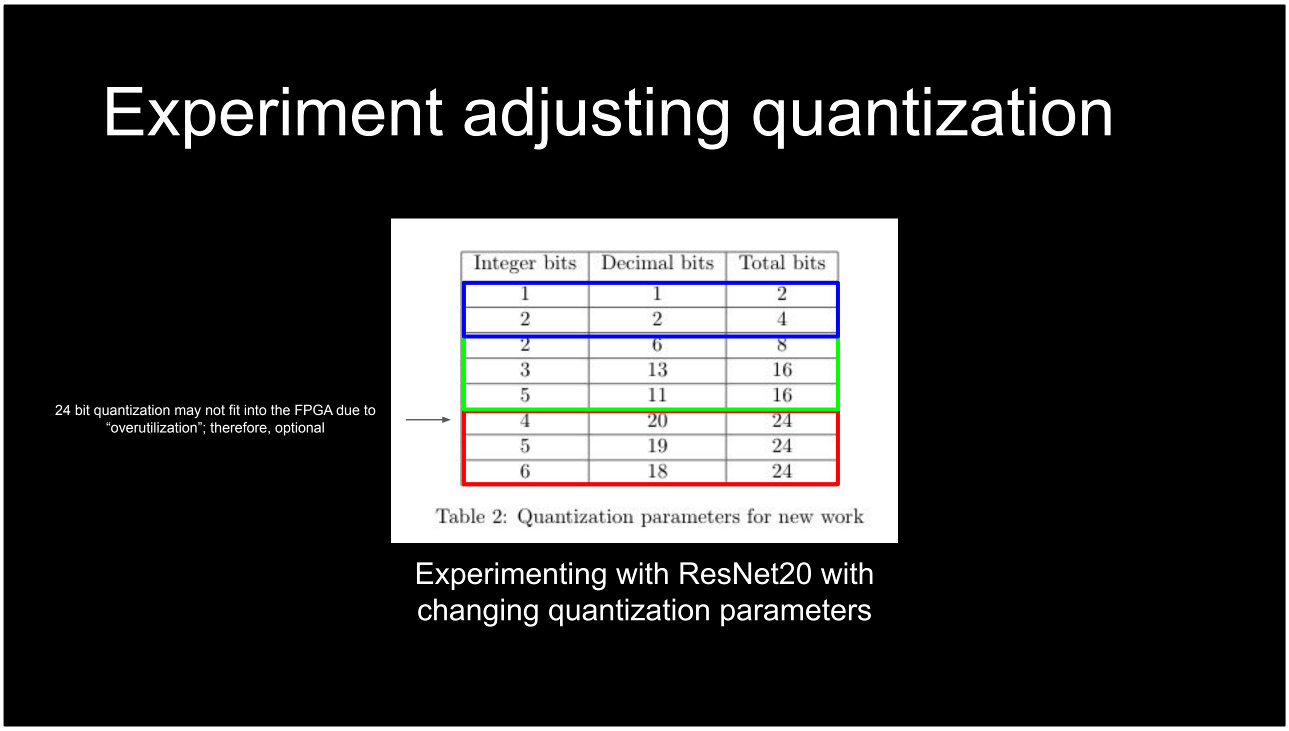
New ResNet quantization

- Using well-know parameters <16,4> <8,3>
- ResNet56
- ResNet110

Integer Bits: 4
Fractional Bits: 12
0110 011100111010
Total Bits: 16

In addition to implementing Skip and testing its effects, we will also be exploring the impact on accuracy of quantization on larger ResNets than what the existing work did. Specifically, the paper tested well known quantization parameters, which you see here on the slide. Recall that quantization takes floating point weights and converts them into fixed point numbers. The first digit means the total number of bits per number, and the second is the number of bits allocated to the integer portion.

We will use these two well known parameters and apply them (along with Skip) to ResNet56 and 110, and convert the resulting models into FPGA bitstreams to get measurements about just how many resources the FPGAs require.



Experiment adjusting quantization

Integer bits	Decimal bits	Total bits
1	1	2
2	2	4
2	6	8
3	13	16
5	11	16
4	20	24
5	19	24
6	18	24

Table 2: Quantization parameters for new work

24 bit quantization may not fit into the FPGA due to "overutilization"; therefore, optional

Experimenting with ResNet20 with changing quantization parameters

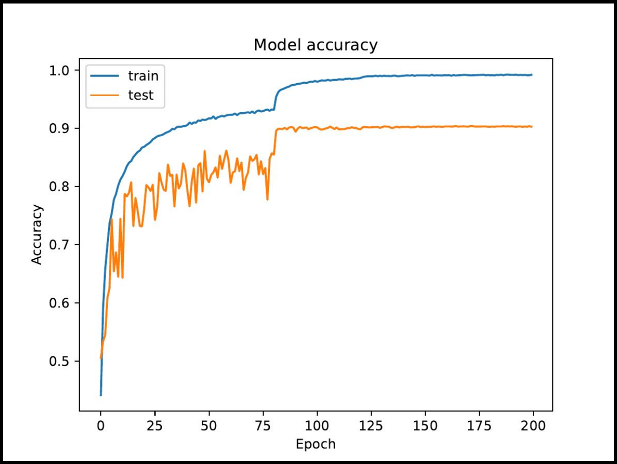
In addition, we also wanted to see the impacts of using some other, not-as-common quantization parameters. For our MVP, we will test ResNet20 with the parameters (in the green box in the slide). Our rationale for selecting these values is that they are close to the "well-known" ones already tested by the paper.

If time permits we want to also test the impacts of (in the blue box). The idea is that with smaller parameters we end up throwing out a lot more information, so running these tests will help us answer the question of "Does it matter?" Is the reduction in accuracy worth the reduction in the size of the FPGA implementation?

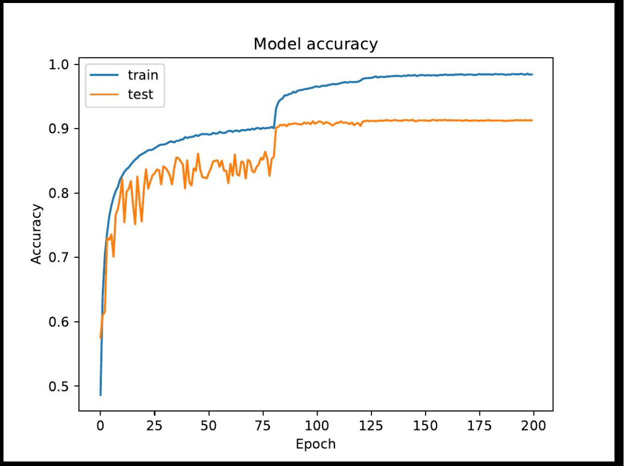
I doubt we'll have the time, but as additional experiments we want to replicate all of these experiments with ResNet56 and ResNet110.

We initially wanted to test quantization values with 24 total bits in size, but due to time constraints and some insight from the data we've already collected, the resulting models might just be too large for the FPGA we are targeting. The goal of the work is to reduce the resource utilization of the models, in any case, and prior work did not use anything larger than 16 total bits.

Accomplishments



ResNet20

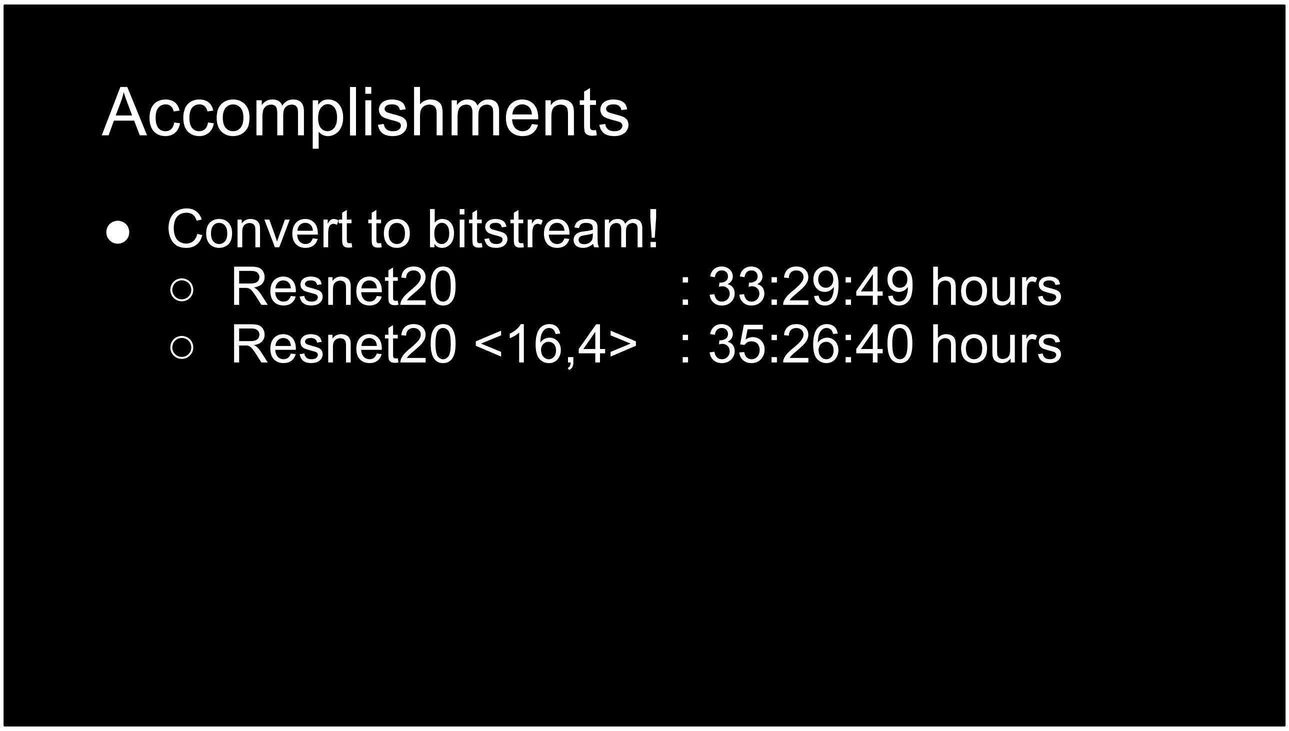


ResNet20 <16,4>

Alright, that's a lot about what we are planning on doing, so what have we done so far?

Our first goal is to replicate the results of the in-review paper. We are almost done with this part.

We have successfully trained ResNet20 on the Cifar10 dataset (left image), and also trained a quantized version of it (right image). Interestingly enough, the accuracy of the quantized one appears marginally better than the non-quantized one. We are using our own computers with Nvidia GPUs to train these networks, which helps train them in around an hour. We are planning on training the networks multiple times to see the average top accuracy and standard deviation.

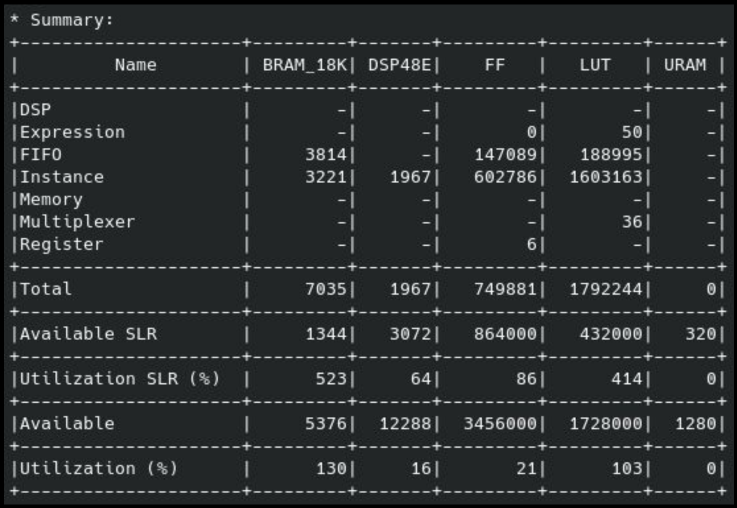


Accomplishments

- Convert to bitstream!
  - Resnet20 : 33:29:49 hours
  - Resnet20 <16,4> : 35:26:40 hours

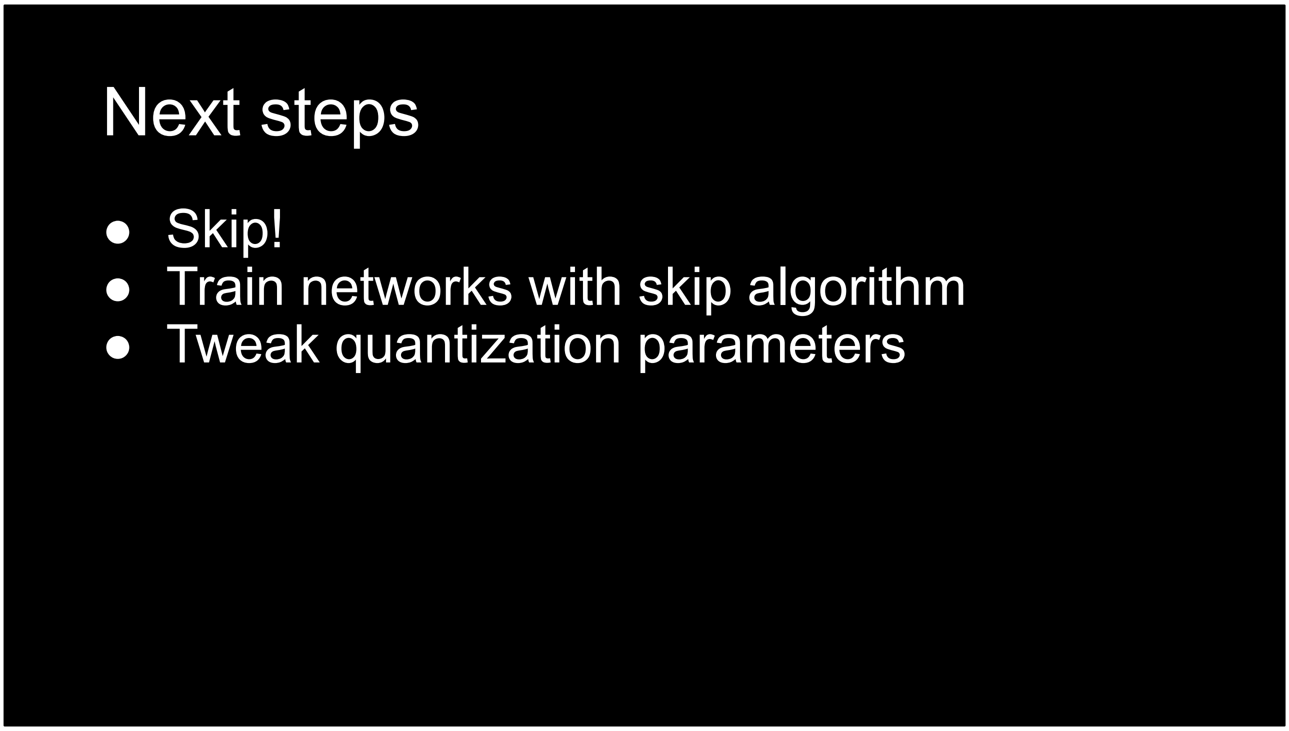
We also converted two of the previously trained models, and what you're seeing on the slide is just how long it took to synthesize the FPGA bitstream, or the program that's actually programmed into the FPGA. We have scaled back some of our initial plans with regards to just how many models we are going to synthesize.

Accomplishments



```
* Summary:
+---------------------+---------+-------+---------+---------+------+
|         Name        | BRAM_18K| DSP48E|    FF   |   LUT   | URAM |
+---------------------+---------+-------+---------+---------+------+
|DSP                  |        -|      -|        -|        -|     -|
|Expression           |        -|      -|        0|       50|     -|
|FIFO                 |     3814|      -|   147089|   188995|     -|
|Instance             |     3221|   1967|   602786|  1603163|     -|
|Memory               |        -|      -|        -|        -|     -|
|Multiplexer          |        -|      -|        -|       36|     -|
|Register             |        -|      -|        6|        -|     -|
+---------------------+---------+-------+---------+---------+------+
|Total                |     7035|   1967|   749881|  1792244|     0|
+---------------------+---------+-------+---------+---------+------+
|Available SLR        |     1344|   3072|   864000|   432000|   320|
+---------------------+---------+-------+---------+---------+------+
|Utilization SLR (%)  |      523|     64|       86|      414|     0|
+---------------------+---------+-------+---------+---------+------+
|Available            |     5376|  12288|  3456000|  1728000|  1280|
+---------------------+---------+-------+---------+---------+------+
|Utilization (%)      |      130|     16|       21|      103|     0|
+---------------------+---------+-------+---------+---------+------+
```

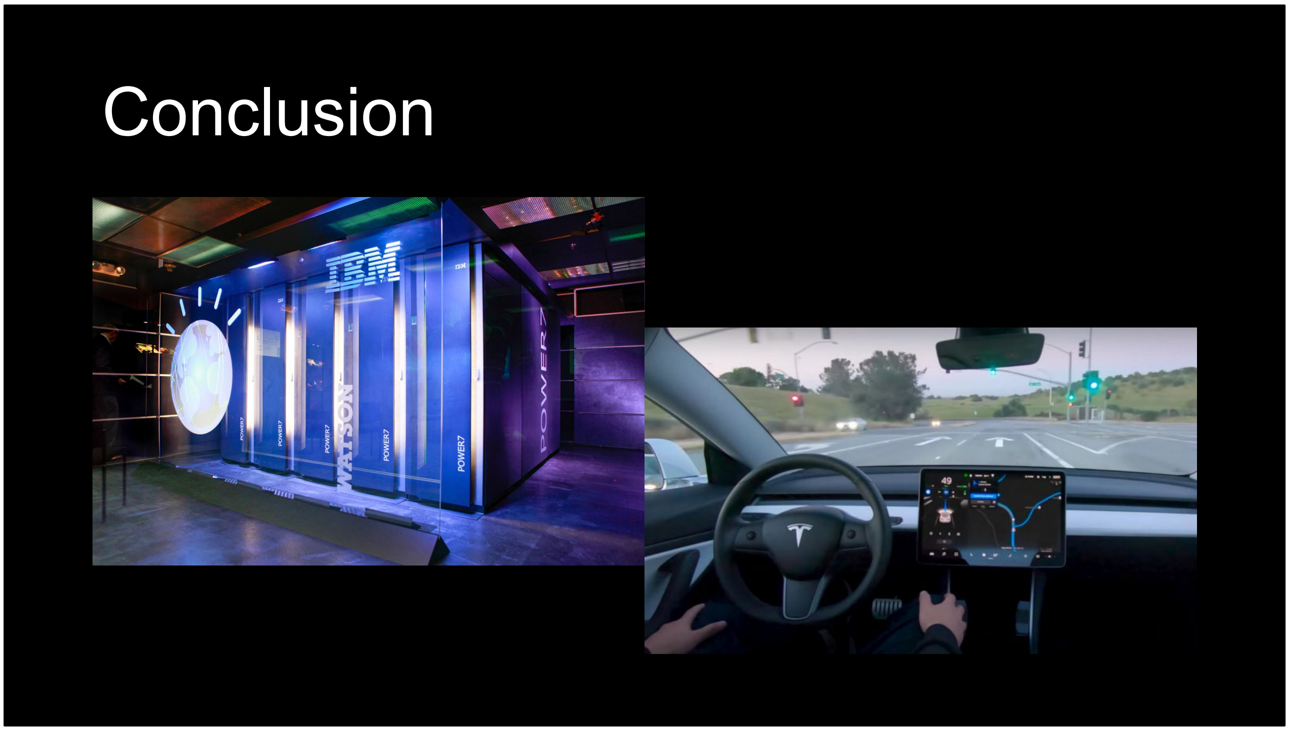
This is a screenshot of the summary of converting a quantized ResNet20 <16,4>. I want to draw your attention to the very last line of the table, where the table summarizes the percent utilization of the FPGA. Specifically, the Block RAM and LUT (think logic gates) utilization is over 100%, so this design would not be able to fit into this specific FPGA being targeted. This is why this research is important-- we *want* to fit these networks in FPGAs! Every littler bit counts!



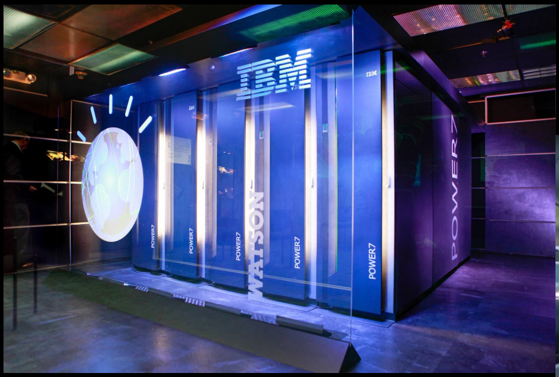
Next steps

- Skip!
- Train networks with skip algorithm
- Tweak quantization parameters

Implement Skip - This uses knowledge distillation, where one trained model teaches another student model how to act, effectively, by having the teacher be a regular ResNet, and the student be ResNet with some skip connections removed. Once we do this, we want to train models using this approach, and then quantize and synthesize those models. Finally, we want to tweak the quantization parameters, as we mentioned before, for all of these new networks, to explore the impact of these different tools on accuracy.



Conclusion



In conclusion, we want to help artificial neural networks to be usable everywhere, and to that end we need to address the current energy and compute requirements to run them. We are continuing work exploring further optimizations to neural networks to make them fit for use with FPGAs, so that these networks may be deployable in more places to enable more complex, on-site analysis and autonomous decision making.

Any questions?