Project 4 (Optional): Distillation and Compilation

Oliver Fowler and Brian Park

North Carolina State University, Computer Science 591/791-025

November 2022

1 Distillation and TVMC Optimizations

For the first part of the project, we applied knowledge distillation on a DNN model [1]. Distillation requires a compressed model to be trained or distill the knowledge of a pretrained, larger model as shown in Figure 1. We are able to leverage NNI to do compression and then use native PyTorch code to do knowledge distillation [2, 3]. For compression techniques, we are already familiar with pruning and quantization. Unfortunately, NNI's quantization is not well supported, as we have seen from a previous project. So for this project, we only applied pruning.

We also tried additional techniques to speed up training and distillation, such as using mutliple GPUs. Fortunately, bridges-2 nodes have DGX-1 system on `GPU-shared` allocations. An allocation up to 4 V100 GPUs connected via NVLink can be used for data parallel training [4]. PyTorch natively supports data parallel training on multi GPUs in an intra-node setting via `torch.nn.DataParallel` and `torch.nn.DistributedDataParallel` [3]. We were inspired to use such tasks to fully utilize the supercomputer hardware from related works [5]. Although we successfully trained in a `DataParallel` and `DistributedDataParallel` setting, we realized this process is impossible to apply at the distillation stage. First, NNI creates wrappers around PyTorch modules, making some of the API incompatible with each other. Once a PyTorch model is converted to a distributed setting, it's hard to convert it back to single GPU setting. Plus, we noticed that there is not much work related to distributed distillation on large models to small models, so we moved on to a simpler setup.

After leveraging NNI and PyTorch to do knowledge distillation to a smaller model, we then used TVM to do end-to-end compiler optimizations to the model [6]. We were able to convert PyTorch to a ONNX file using PyTorch's API. Then we convert the ONNX file to an executable using TVM.

2 DNN Model

For this project, we chose ResNet-101 that we previously trained from a previous project on the CIFAR-10 dataset [7, 8]. Given the time constraint, we didn't want to train a large model from scratch. We actually tried to use DenseNet initially, but ran into issues related to OOM and training speed since we have to load both the teacher and the student model [9]. We also tried to skip training from scratch by using the ImageNet dataset, since Torchvision provides pretrained models for it [10, 11]. But turns out ImageNet is a very large dataset and it would take days or weeks even on state of the art hardware. We tried some other techniques to utilize the powerful resource of bridges-2 supercomputer, but later realized that NNI and PyTorch makes this process incompatible [12].

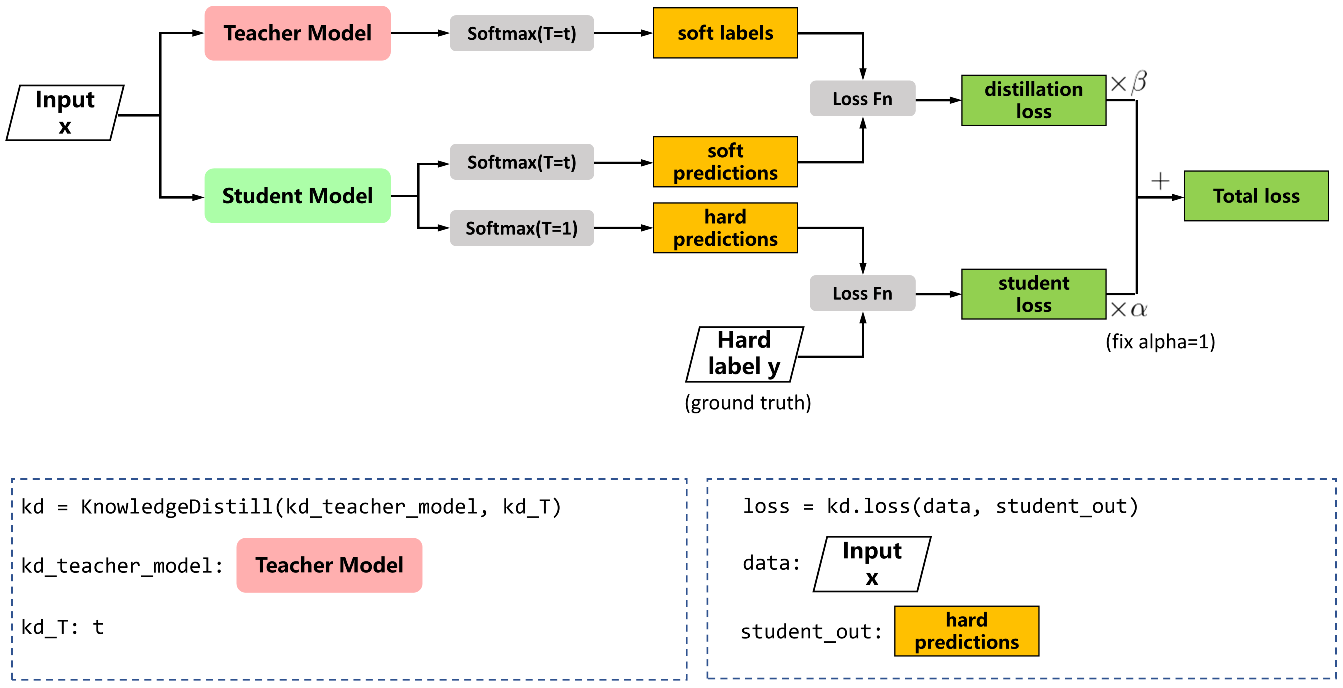
We chose to distill ResNet-101 model to ResNet-18 model that is also pruned via NNI. We did not apply quantization since NNI doesn't support it properly as we have observed from a previous project.

The ResNet-18 model is structured pruned via NNI with L1NormPruner with a sparsity of 10% on only `Conv2D` layers [13]. We could not apply pruning on `Linear` layers as the distillation process becomes broken due to incompatible tensor sizes.

Since there is extra hyperparameter of temperature in knowledge distillation, so we tried temperatures of $T = 1, 5, 10, 15, 20$, just like how Hinton experimented between values of T from 1 to 20 [1]. We train and finetune the weights for 20 more epochs.

After distilling in PyTorch, we converted it to ONNX model files and then to TVM compiled executables. Unfortunately, we couldn't get TVM to compile to a GPU target due to an unknown issue with CUDA libraries not being able to be found and linked properly. Instead, we compiled to LLVM target and benchmarked on the CPU for a fair comparison across optimizations.

For TVM, we also applied tuning that uses XGBoost to automatically tune the performance of the TVM executable [14].



```
kd = KnowledgeDistill(kd_teacher_model, kd_T)

kd_teacher_model: Teacher Model

kd_T: t
```

```
loss = kd.loss(data, student_out)
```

Figure 1: Knowledge Distillation Pipeline [2]

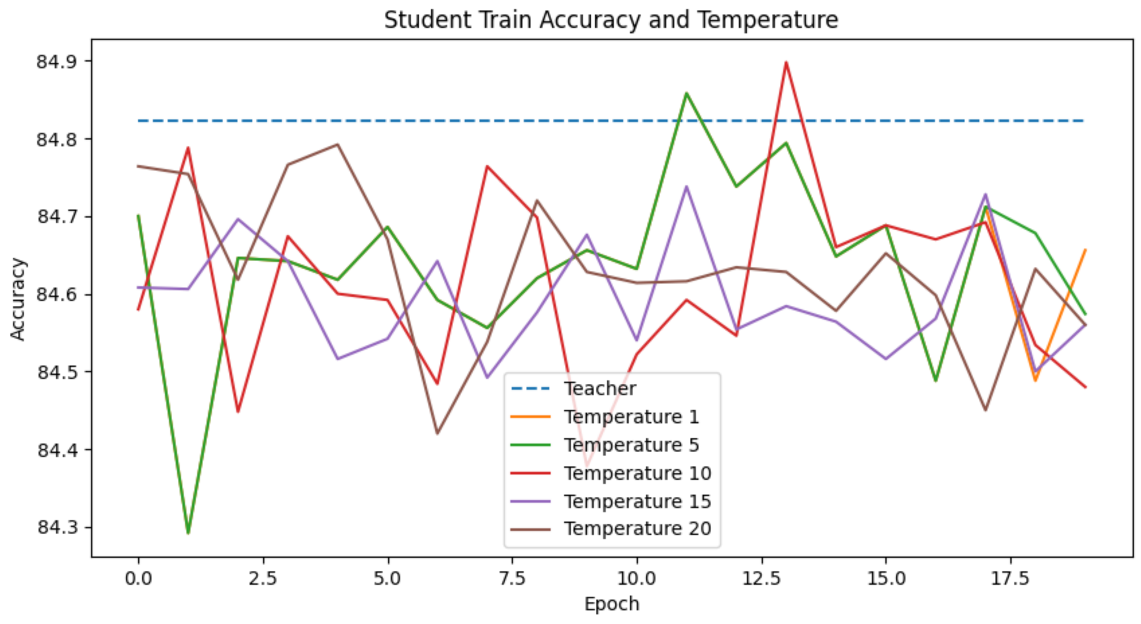


Figure 2: Knowledge Distillation Training Accuracy over Epochs

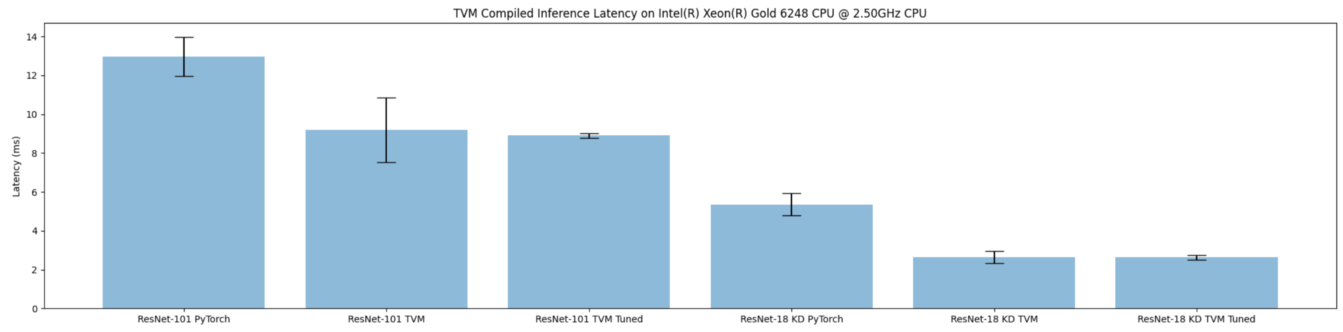


Figure 3: PyTorch vs TVM Benchmarks

3 Experimental Results

The original ResNet-101 model, or teacher model has a training accuracy of 84.822% and a validation accuracy of 79.27%. When applying distillation, we got a training accuracy of 84.6& on the ResNet-18 distilled model. In the end, we chose the distilled model with a temperature of 1 since it had marginally best training accuracy. Figure 2 shows the training accuracy over epochs. It makes sense that the distilled model should not outperform the teacher model. Unfortunately, changing the temperature doesn't vary a lot, and it's also suspicious why the training accuracy starts off already high. Compared to related works where Stanton experiments and questions the knowledge distillation works, it seems that we may have a flaw in our experimental methodology [15].

Figure 3 shows the benchmarks of each framework on the CPU of bridges-2, which has a Intel(R) Xeon(R) Gold 6248 CPU. We took one sample image from CIFAR-10. From PyTorch to TVM tuned model, we decreased inference time from 12.97 ms to 8.90 ms for ResNet-101 alone. For ResNet-18 with knowledge distillation the inference time decreased from 5.35 to 2.64. We see that 2-3 ms is saved from just TVM.

The model size doesn't change at all between compilation phases, since we aren't modifying the weights in the compile phase. But interestingly, we see that for ResNet-101 the model size is 170 MB, while the distilled smaller model is 41 MB. At least with knowledge distillation, we're able to retain accuracy within a percent, as well as gain 4.14× size reduction and 4.95× speedup on inference time with compilation optimizations. If this was video processing application where images are streamed, we'd increase processing performance from 77 FPS to 379 FPS.

4 Lessons Learned

Since we had much powerful compute resources, we took this as a learning opportunity to learn how to utilize it for deep learning. The number one issue we faced is waiting. Since there was only over a week to complete this assignment, we engineered how to make workflow quicker. We subsampled our dataset as well as used multiple GPUs for data parallelism. We can clearly see how deep learning workloads take days or months to train. Fortunately, we made sure to start early.

We also learned how to use TVM. Overall it's very simple to use. If we had more time, we wish we figures out how to properly link CUDA libraries and compile to a GPU target for even more speedup.

Overall, we learned how to do a full stack integration with NNI and TVM for model compression and compilation. We started with a large model and optimized it for a target, which was the V100 GPU in our case. The tools really helped in easing the workflow, as the whole pipeline takes few lines of code and is really simple to program.

5 GitHub Repository

The GitHub repository for this report is publicly accessible here [16]. To reproduce our findings, please read the `README.md` under the `proj4` directory. If there are any setup issues on bridges-2 supercomputer or locally, please contact bcpark@ncsu.edu.

References

[1] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015.

[2] Microsoft, “Neural Network Intelligence,” 1 2021.

[3] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[4] “Nvidia v100 datasheet.” `https://images.nvidia.com/content/technologies/volta/pdf/volta-v100-datasheet-update-us-1165301-r5.pdf`.

[5] Y. You, Z. Zhang, C. Hsieh, and J. Demmel, “100-epoch imagenet training with alexnet in 24 minutes,” *CoRR*, vol. abs/1709.05011, 2017.

[6] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, M. Cowan, H. Shen, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, “Tvm: An automated end-to-end optimizing compiler for deep learning,” in *Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation*, OSDI'18, (USA), p. 579–594, USENIX Association, 2018.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[8] A. Krizhevsky, “Learning multiple layers of features from tiny images,” pp. 32–33, 2009.

[9] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” 2016.

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, “ImageNet Large Scale Visual Recognition Challenge,” *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[11] S. Marcel and Y. Rodriguez, “Torchvision the machine-vision package of torch,” in *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, (New York, NY, USA), p. 1485–1488, Association for Computing Machinery, 2010.

[12] S. T. Brown, P. Buitrago, E. Hanna, S. Sanielevici, R. Scibek, and N. A. Nystrom, “Bridges-2: A platform for rapidly-evolving and data intensive research,” in *Practice and Experience in Advanced Research Computing*, PEARC '21, (New York, NY, USA), Association for Computing Machinery, 2021.

[13] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” 2016.

[14] T. Chen and C. Guestrin, “Xgboost: A scalable tree boosting system,” *CoRR*, vol. abs/1603.02754, 2016.

[15] S. Stanton, P. Izmailov, P. Kirichenko, A. A. Alemi, and A. G. Wilson, “Does knowledge distillation really work?,” *CoRR*, vol. abs/2106.05945, 2021.

[16] “Csc 791-025 github repository.” `https://github.com/briancpark/csc791-025/tree/main/proj4`.