BENCHMARKING OF DNN COMPRESSION TECHNIQUES

Aman Singh Thakur / Sakshi / Astha Baranwal

amansinghtha@umass.edu, fsakshi@umass.edu, abaranwal@umass.edu

ABSTRACT

Using multiple hidden layers and over-parameterization, Deep Neural Networks have recently become popular due to their ability to fit a vast array of large datasets. In this era of data explosion, however, the resultant large size poses challenges for running and evaluating these models on devices with limited memory and computation power. In this project, we explore and compare the performance of different DNN compression techniques like Network Pruning, Quantization, and Knowledge Transfer across various benchmarks like Accuracy, Size, and Latency.

1 INTRODUCTION

Here, we discuss the motivation of the project and our hypothesis or Research outcomes.

1.1 Why compress Deep Neural Network ?

With the advent of the Internet of Things, we have an opportunity to deploy trained DNN models on every device on the planet. However, large DNN models create a problem for IoT devices with limited memory and small processing power. For example, the famous GPT-3 DNN model has 175 billion parameters and cannot be deployed on any small device. To fully utilize the benefits of DNN, we must learn to compress these large models into smaller footprints, democratizing everyone to train and deploy them.

Apart from hosting these large models on edge devices, compression of deep neural networks will help us optimize over-parameterization in large DNNs. As described in paper (Mishra et al., 2020), it's worth mentioning that compression of DNN allows for better storage capacity, fewer computation requirements, earliness, more privacy, and less energy consumption.

1.2 Hypothesis or Research outcomes

As part of this project, we will train the baseline ResNet-18 model on an image classification dataset, CIFAR-10. Using techniques like Network Pruning, Quantization, and Knowledge Transfer, we will compress the baseline and benchmark their performance across evaluation metrics like accuracy, size, and latency.

2 BACKGROUND

2.1 Sparse Representation In Neural Network

Large Deep Neural Networks produce state-of-the-art accuracies at the cost of over-parameterization. Limiting the network to its sparse representation to avoid the same laid the foundations of the Network pruning technique, and it is flavored. One of the flavors of network pruning is Soft Filter Pruning (SFP). As mentioned in article (Gu et al., 2021), SFP enables the pruned filters to be updated when training the model after pruning. It follows the L2 norm of the weights criterion and zeroes out filters without removal until the end.

Another popular approach for Network Pruning is weight Pruning (Carreira-Perpinan & Idelbayev, 2018), where we take away the weights least important or valued to the network. Doing so makes the network sparse, less over-parameterized, and drastically reduces size. Weight filtering also allows DNN to be more robust and generalizable.

2.2 Quantization in a Neural Network

Quantization has been used for ages to map input values in a large set to output values in a small set, and it is heavily used in calculus. Quantization usually involves rounding and truncation Quantization in neural networks has been used for a while now. Firstly, training of neural networks is computationally intensive. Efficient representation of numerical values is really important. Secondly, almost all recent neural network models are highly over-parameterized, so there we can reduce bit precision without significant impacts to model accuracy. However, neural networks are extremely robust to aggressive quantization due to over-parameterization. This directly affects how well-posed our problems are, whether we are interested in forward or back-

ward error, etc. So if we are interested in forward error metric (based on prediction quality, complexity, etc.), due to the over-parameterization, there are several varied models types that either approximately or quite exactly optimize this metric. As a result, it is possible to get very good generalization performance with a high error/distance between a quantized model. This is the factor that has been driving force for all the research on novel quantization techniques for neural networks, as mentioned by ((Gholami & Dong, 2021)).

2.3 Distilling Knowledge in a Neural Network

Human civilization has evolved because of our ability to learn from the knowledge passed onto us by our ancestors. Even in our current generation, we can see almost every child has undergone all their learning under this teacher-student framework. Under the direction of a teacher, students can better understand patterns and train themselves to commit fewer mistakes. This real-world approach has led to one of the more popular papers on knowledge distillation (Hinton et al., 2015), which talks about transposing knowledge from a larger pre-trained state-of-the-art deep neural network to a smaller student network for the same dataset. While training a regular student network with a standard cross-entropy loss function, we create an objective function that reflects the user's true objective as closely as possible. As per the previous paper, we can modify the loss function to allow the objective function to generalize well to the new data, similar to how the parent model has generalized. Other related papers and articles like (Cho & Hariharan, 2019), (Gou et al., 2021), and (Mishra et al., 2020), etc., have echoed this strategy with slight modifications.

One of the biggest challenges of Knowledge Distillation is that we are still limited to training a teacher model, which will be extremely computationally expensive.

3 Approach

- Firstly, we trained the baseline ResNet-18 model on the CIFAR-10 dataset and described the training model parameters, layers, and baseline metrics.
- With the ground truth model established, we described and performed each compression technique and explained the change in model parameters for each technique.
- Lastly, we used multiple evaluation metrics (accuracy, latency, and compression%) to benchmark the performance of different compression techniques and understand the best techniques suited for different use cases.

4 Experiment

4.1 The Data

To effectively evaluate DNN compression techniques, we need a dataset with multiple DNN layers to train it. Earlier, we planned to use a subset of ImageNet data (ILSVRC) for training complex neural networks until we encountered issues due to limited computational bandwidth. We are now using the Canadian Insitute for Advanced Research (CIFAR-10) dataset, which contains over 60,000 32x32 images in 10 different classes. With roughly 170MB in size, this dataset is perfect for us to perform proof of concept experiments and later extrapolate them to larger datasets. The training and validation images are annotated and stored in XML files. We have parsed these annotations using the PASCAL development toolkit. Grouped by their synsets, All annotated images (training, testing, and validation) will be available in the same folder. We have used PyTorch to transform (resize, rotate, etc..) and load the image dataset.

4.2 Using CIFAR instead of Imagenet

With more than 14 million hand-annotated images and 20,000 categories, ImageNet is one of the biggest and wildly used datasets for image classification problems. However, it would take us days to train the ResNet model on the ImageNet dataset unless we have access to massive distributed computational power. To prove the DNN compression techniques' success, we will train the ResNet models against a much smaller dataset, CIFAR-10. In our experiments, we have noticed that training the ResNet model on CIFAR-10 takes roughly 30-40 mins for ten epochs.

4.3 Data pre-processing and Base Model Training

We have resized images to 224 with ResNet's normalization to adjust to computation limitations. Other transformations like normalization, random horizontal flips, and rotations are also added to the dataset. Data augmentations are great for increasing training set and testing set sizes and regularizing the neural network more, thus forcing it to be more flexible in predictions. We split the training set into training and validation sets. The baseline has an accuracy of around 94% and model size of around 42.621 MB, and a prediction latency of 2.27 ms.

Next, a pre-trained ResNet18 model is deployed as a base. The learning rate scheduler is added to decay the learning rate gradually. All layers are trained with Adam optimizer to increase accuracy. The backpropagation technique on the cross-entropy loss function is used to update the parameters (tensors) of the model using gradients. The model is trained using Cuda GPU provisions in Google Colab. We saved the model after training and then calculated the baseline model's size and its accuracy on the test set.

4.4 Pruning

Pruning allows us to compress the model size with marginal accuracy loss, as most models are over-parametrized. As shown in Figure 1, Pruning eliminates the weights with a low magnitude that do not contribute much to the final model performance. Till now, we have understood the pruning strategy, which follows the bit mask method to determine the weights that should be used for training the model. PyTorch offers specific APIs to apply for pruning which includes global unstructured pruning, local structured pruning, and local unstructured pruning to tensors randomly, by magnitude, or by a custom metric. We have done experiments implementing local unstructured pruning and random unstructured pruning with different pruning percentages of the convolution layer weights.

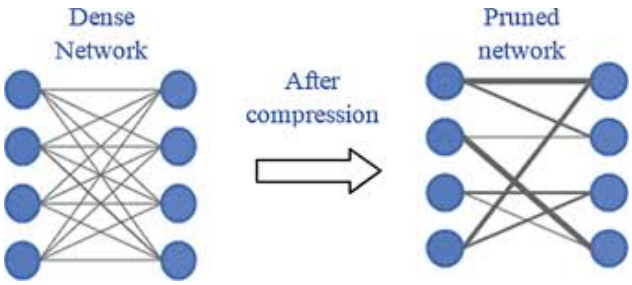

Figure 1. Neural Network Pruning

In L1-Unstructed pruning, the least valued model weights are pruned. We did not observe much variance in the accuracy till 60% of the model was pruned, as shown in Figure 2. This signifies the "overparameterization" of state-of-the-art deep neural networks. This means that the model shows no significant drop in accuracy even when 60% of the weights are pruned and sharply falls only after pruning it more than 60%. However, we couldn't observe a significant change in latency. We are using PyTorch, which only masks the values of the least-valued weights without actually removing them. We also experimented with random unstructured pruning and observed that the accuracy is lower than the l1-unstructured pruning as random pruning masks the weights randomly.

4.5 Quantization

Quantization converts biases, weights and activations of tensors with reduced precision (8 bit INTs or INT8) instead of full precision (32 floating points or FP32). Less memory and computation power is needed on such neural networks. PyTorch supports static and dynamic quantizations. Static quantization quantizes weights and activations ahead of time. Dataset calibration is required post training. Dynamic quantization quantizes weights ahead of time but the activations are quantized on the fly (online). Since PyTorch's

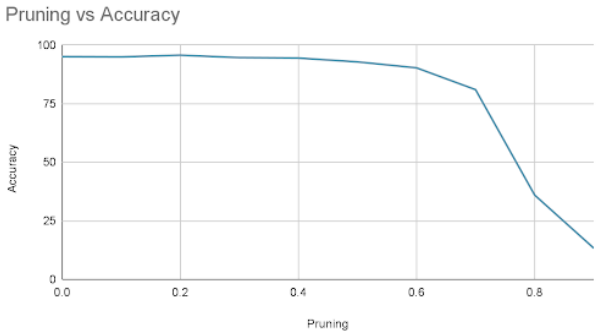

Figure 2. Pruning vs Accuracy

quantization modules do not yet have GPU support, and CPU computations take incredibly long amount of time, we decided to calculate our evaluation metrics and train the quantization model using some batches of CIFAR10 dataset over multiple epochs.

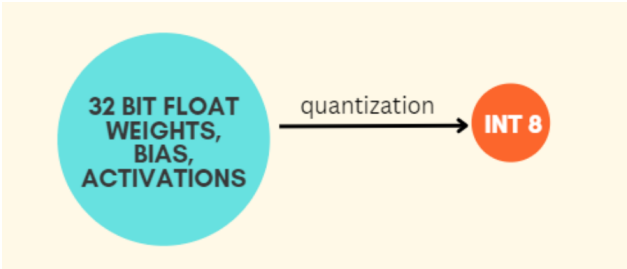

Figure 3. the concept of quantization

Dynamic quantization model gave an accuracy of approximately 94% on 1280 train data points in each epoch (15 epochs). The accuracy more or less remained the same possibly because instead of using the complete training dataset, we are using some batches of the dataset only. Regardless, there is not much quantization done in the dynamic quantization model as ResNet18 is composed of mainly Convolution layers. Pytorch's dynamic quantization does not yet support Convolutional layers, but it does support Linear layers. So only a little difference was observed in model size as only the last output layer is Linear layer.

Static quantization gave train and test accuracy of 88-89%. The static quantized model size (10.79 MB) got reduced to approximately one fourth of the baseline model's size (42.62 MB), which was expected because quantization converted the bias, weights and activations from 32 point floats to 8 bit INTs. Static quantization model gave an accuracy of around 88%, which is lower than the baseline's accuracy. This was to be expected as we are decreasing precision of

the entire model. Dynamic quantization's model size didn't have much change as our baseline model was trained on ResNet18, which had convolutional layers. We have not calculated latency for quantization as there is no support for GPU to calculate prediction latency for quantized model.

4.6 Knowledge Distillation

To perform our experiment, we must first understand the intricacies of choosing a student network. If we consider a student with more hidden layers, we will experience better accuracy, but the model will not be considerably smaller than the teacher network. While taking a student network with fewer hidden layers, the student will underfit and produce poorer accuracy. On the other hand, a smaller network will be more transparent and quicker to evaluate.

Inspired from (Li) work on testing knowledge distillation on the Res-Net-18 model, we chose to consider a five-layer Convolutional Neural Network (CNN). With respect to the 18-layer ResNet-18 model, we expect the student model (5-layer CNN) to perform less accurately, but the model is only 12% of the teacher's size.

To simulate a similar training environment, we train the student network using the same pipelines built for training the teacher model. After being trained on cross-entropy loss for ten epochs, we can see that the student is performing at an accuracy of 66.075 % while the teacher performed 94.24%.

In Knowledge distillation, the teacher distills the knowledge of the predicted class probabilities to the student as "soft targets" while the student tries to minimize the relative entropy between target values and predicts values from raw images using KL-Divergence D_{kl}. We use hyperparameters T (temperature) and (relative importance) to regularize the training process. Using T, we can produce a suitable set of soft targets for the student model, and provides the relative importance of hard targets (teacher's predictions) to the soft targets. We use hyperparameters T (temperature) and (relative importance) to regularize the training process. Using T, we can produce a suitable set of soft targets for the student model, and provides the relative importance of hard targets (teacher's predictions) to the soft targets. Figure 5 captures a simple representation of Knowledge Distillation is captured in paper (Gou et al., 2021).

As mentioned in (Hinton et al., 2015), The equation in Figure 4 represents the resultant loss function used to train the student networking using Knowledge Distillation. Here P are the soft labels from the teacher network, and Q are the students' softmax scores.

Here P are the soft labels from the teacher network and Q is the softmax scores from the student.

$$\frac{1}{m}\sum_{j=0}^{m}\left\{2T^2\alpha D_{KL}\left(P^{(j)},Q^{(j)}\right)-(1-\alpha)\sum_{i=1}^{c}y_i^{(j)}log\left(1-\widehat{y}_i^{(j)}\right)\right\}$$

Figure 4. Knowledge Distillation Loss Function

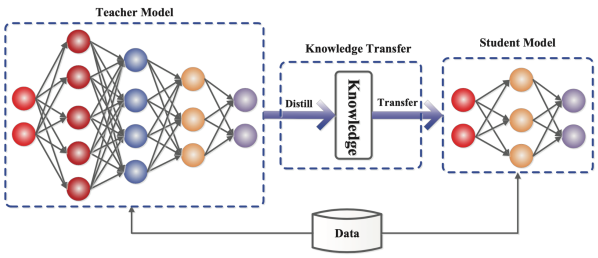

Figure 5. Visualizing Knowledge Distillation

As is evident from Figure 6, we can see that the student trained using Knowledge distillation with normal parameters performs poorly compared to the raw model's accuracy. After performing KD with Temperature (T) =1, we place a very high weightage for the teacher's predicted output. Placing such a high weightage means that the relative entropy loss will be large initially, and it will take longer to converge. However, After training the same model with Temperature (T)=0.01, we let the student train under the teacher's mild influence, and we immediately observed better performance when compared with the raw KD model. We see that with hyper-parameter tuning, the model finally outperforms the raw student's accuracy of 66.075 % with 67.7%. After observing the trends in the graph, we can see that Knowledge distillation with hyper-parameter tuning will perform with better accuracy if trained for more epochs.

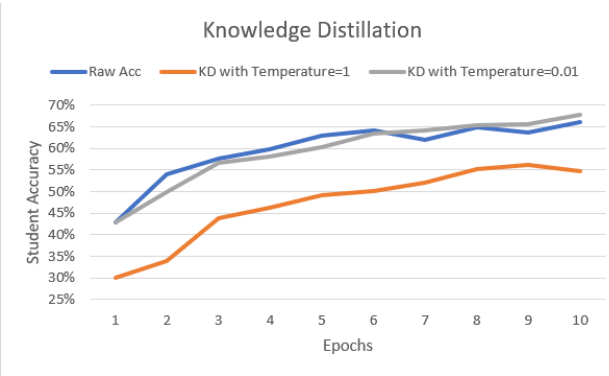

Figure 6. Knowledge Distillation for Student network

4.7 Evaluation Metrics

There are several evaluation metrics we will be considering to fully benchmark and analyze the models we obtain after applying DNN compression techniques (pruning, quantization and knowledge distillation (KD)) on the original DNN model. 1. Model compression measure, 2. accuracy, 3. latency are perfect as evaluation metrics for benchmarking DNN compression techniques (see Table 1). Although latency is difficult to measure for quantization model because there is no GPU support for PyTorch quantization modules yet.

- Model Compression Measure (%): Model compression measure is the ratio of compression of the quantized model's size to baseline model's size. Model size is calculated in MB and it is dependent on the sizes of the parameters of a neural network like biases, weights, activation functions, neurons input layer and output layers. Compression is really important for huge neural networks as these models have to run n a myriad of IoT devices which might not be able to handle massive models due to storage or computation limits.
- Accuracy: Accuracy is the fraction of correct predictions done by that model. Usually we end up losing accuracy when we compress a neural network model. The goal is always to compress the neural network in such a way that we lose marginal or really less accuracy.
- Model Prediction Latency: Prediction latency is the amount of time elapsed in predicting a new data point. We expect this metric to fall for the compressed models as they are expected to be faster than the baseline model. Cuda event recorder is used to measure the elapsed time to calculate latency in milliseconds.

Table 1. Evaluation Metrics to Benchmark DNN Compression Techniques

EVALUATION METRIC	BASE	PRUNING	ST.QUANT.	KD
MODEL SIZE	42.69MB	17.08MB	10.79MB	5.19MB
COMPRESSION MEASURE %	0%	60%	74.72%	87.84%
ACCURACY %	94.24%	90.26%	88.44%	67.7%
LATENCY	2.27MS	2.05MS	N/A	2.49MS

5 CONCLUSION

In this project, we benchmarked the deep neural network compression technique with quantization, pruning, and knowledge distillation techniques. We observed all the models for their accuracy, latency, and model size (model compression factor). There were slight changes in all three metrics in all three DNN compression technique models from the baseline model. The accuracy decreased in all the compression techniques when compared to the baseline model's accuracy. Pruning performed the best, followed by static quantization and then knowledge distillation. Pruning still lost some accuracy of the baseline model as the tree will not learn optimal parameters. With quantization, the neural network loses quite a lot of its precision, and hence, the accuracy is lower too. In knowledge distillation, everything depends on Student Size. Since the student is 5-layer CNN with respect to the 18-layer ResNet-18 model, the accuracy will always be lower than the raw model. However, our hypothesis for knowledge distillation is that we will use the raw Student model and train it with knowledge distillation and attain better accuracy So, raw Student is 66% but with the knowledge distillation technique, we get 67%. So we achieve compression of DNN in a way that we can use the raw model as a reference and train smaller models to yield better accuracy. The model size of the pruning model got reduced to 40% of the baseline model. The model size of the static quantization model got reduced to 25% of the baseline model, which was definitely as expected because we converted 32 float points to 8-bit integers. The prediction latency of pruning did improve when compared with the baseline model, but not by a lot. Latency is difficult to compute for quantization as GPU support is not there yet for quantization modules. Knowledge distillation's latency is more than that of the baseline, which is intriguing and we can explore it as future work.

In future work, we can also benchmark the compression techniques with more metrics like FLOPs (floating point operations per second) to measure the performance of the models, computation time, and computation resources used by the models.

6 TEAM CONTRIBUTIONS

- Aman Singh Thakur - Literature Survey, Baseline Training, Performing Knowledge Distillation, Running Evaluation Metrics, Hyper-parameter Tuning
- Astha Baranwal - Literature Survey, Baseline Training, Performing Quantization, Running Evaluation Metrics, Exploring Quantization flavors
- Sakshi - Literature Survey, Baseline Training, Performing Network Pruning, Running Evaluation Metrics, Exploring Network Pruning flavors

REFERENCES

Carreira-Perpinan, M. A. and Idelbayev, Y. "learning-compression" algorithms for neural net pruning. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 8532–8541, 2018. doi: 10.1109/CVPR.2018.00890.

Cho, J. H. and Hariharan, B. On the efficacy of knowledge distillation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

Gholami, A. and Dong, Z. A survey of quantization methods for efficient neural network inference. Technical report, Computer Science Department, UC Berkeley, 2021.

Gou, J., Yu, B., Maybank, S. J., and Tao, D. Knowledge distillation: A survey. *International Journal of Computer Vision*, 2021.

Gu, S., Feng, Y., and Xie, W. Pruning-then-expanding model for domain adaptation of neural machine translation. *CoRR*, abs/2103.13678, 2021. URL https://arxiv.org/abs/2103.13678.

Hinton, G., Vinyals, O., and Dean, J. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015. URL http://arxiv.org/abs/1503.02531.

Li, H. Exploring knowledge distillation of deep neural.

Mishra, R., Gupta, H. P., and Dutta, T. A survey on deep neural network compression: Challenges, overview, and solutions. *CoRR*, abs/2010.03954, 2020. URL https://arxiv.org/abs/2010.03954.