Efficient Knowledge Distillation for Conversion Prediction Models

Alex Racapé
Dept. of Computer Science
Columbia University
New York, NY
abr2184@columbia.edu

Kimberly Collins
Dept. of Computer Science
Columbia University
New York, NY
kgc2138@columbia.edu

Mahdi Tabesh
Dept. of Electrical Engineering
Columbia University
New York, NY
mt3846@columbia.edu

Rohan Singh
Dept. of Computer Science
Columbia University
New York, NY
rs4607@columbia.edu

***Abstract*—Within Google Ads, conversion prediction models play a central role in auction bidding, yet operate under strict constraints on serving latency, memory footprint, and infrastructure cost. Although larger teacher models deliver strong predictive performance, their computational overhead makes them unsuitable for real-time deployment at Google's scale. In this project, we investigate model-compression techniques to develop efficient student models that preserve accuracy while reducing inference cost. We experiment with optimizations around knowledge distillation and quantization within Google's Pyplan-based production pipeline, which runs on Pufferlite TPUs. Our experiments demonstrate that a distilled and quantized student model can achieve a Poisson log loss of –0.15% over existing models in production, while improving critical-path serving latency by 0.14 ms. Preliminary experiments on online traffic further show consistent gains in advertiser spend and desired spend, confirming that our optimization pipeline yields both algorithmic and business-level improvements. These results illustrate that carefully designed knowledge-distilled student models can meet stringent real-time constraints without sacrificing predictive accuracy, making them strong candidates for scalable production deployment.**

***Index Terms*—knowledge distillation, quantization, optimization, conversion prediction**

I. Introduction

A. Problem Overview

Google search and shopping queries produce ranked search results which are ordered based on "bids" placed by various companies. These bids are calculated based on the predicted conversion rate (pConvs) or predicted conversion value (pValue) on a click level depending on the bidding strategy used by the customer. A conversion is a desired outcome which is defined by the site administrators, and this could refer to a shopping purchase, newsletter subscription, or app download depending on the context. The expectation values:

$$\mathbb{E}(\text{Conversions} \mid \text{Click})$$

$$\mathbb{E}(\text{Conversion Value} \mid \text{Click})$$

are predicted by the two models. By accurately predicting pConvs and pValue, downstream bidding and ranking algorithms can better serve advertisers.

Due to high traffic and the use of other downstream bidding algorithms, these models have a strict constraint on latency and model size. Larger models generally attain higher accuracies as the model's capacity increases. However, these larger models cannot be deployed to production due to high latency and serving costs. To address these challenges, this project applies techniques such as knowledge distillation and quantization to reduce latency while maintaining comparable predictive performance. For the purpose of this project we will be focusing on the predicted conversion rate (pConvs) model to iterate over different knowledge distillation techniques.

Within predicted conversions, there is an important distinction between modeled and observed conversions. While many conversions are observed directly by Google in settings like Chrome, Android devices, and others, many conversions may take place on other browsers. In order to accurately predict total conversions, a system also needs to factor in these "modeled" conversions. This is significant because the current models deployed in production have no way of incorporating this information.

B. Architecture Overview

The core of our architecture consists of a student model designed for low-latency production inference, and a larger teacher model used for offline training via knowledge distillation.

The student model uses a single-tower serving architecture and make a single serving prediction with a deep neural network architecture on top of an embedding table. There are additional auxiliary heads that provide valuable information for the model through their losses. The training label for the main head is the total conversions per click which is the sum of observed (reported) conversions and modeled conversions (unreported conversions from non-Google browsers). The specifics of the architecture are proprietary, but our architecture closely follows the design of other deployed pConv models.

Since there could be a long delay between a click and a conversion, i.e. a user could browse a link before making a purchase, the model trains on 0-2 day, 2-7 day, and 7-90 day conversion labels. Each of these windows is represented by a separate prediction head on top of the main network. The final conversion prediction is the sum of all these time-based predictions, expressed as the following:

$$\mathbb{E}[\textbf{conversion}|\text{click}] = \mathbb{E}[\text{convs}_{0-2}]+\mathbb{E}[\text{convs}_{2-7}]+\mathbb{E}[\text{convs}_{7-90}]$$

Since conversions for a given click are best represented by a Poisson distribution, the loss function used for training the model is a Poisson log loss.

The teacher model is about three times the size of the student model, and is trained at a 40 hour delay. It specializes in producing modeled conversion predictions which can be added to the observed conversions to yield the true label.

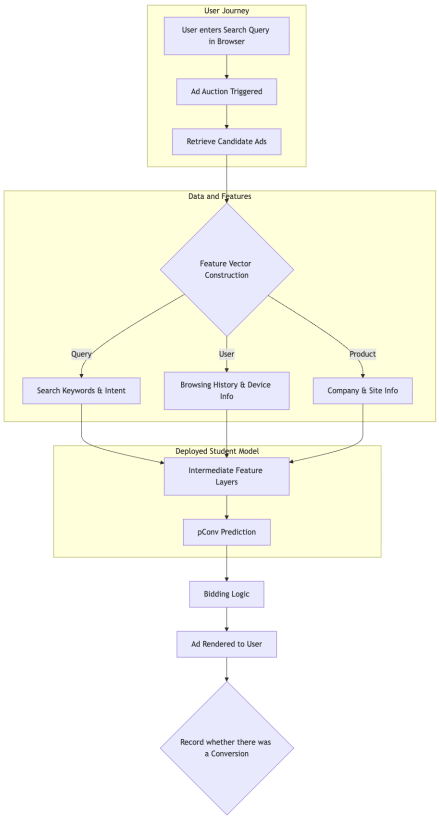

Fig. 1. Model Usage in Production

II. Literature Review

A. Knowledge Distillation

Knowledge Distillation (KD) is a model compression technique which aims to transfer the learned knowledge from a large, high-performing "Teacher" model to a smaller, more computationally efficient "Student" model. The core idea is that the class probabilities generated by the Teacher model, often called soft targets, contain significantly more information about the data than the simple ground truth labels or hard targets. Training the student model to mimic these soft targets allows it to achieve performance comparable to the Teacher, despite having significantly fewer parameters. The concept was initially explored in the context of transferring knowledge from an ensemble to a single model [1]. However, the modern paradigm of knowledge distillation was formalized and popularized by Hinton et al. [2], who introduced the use of temperature scaling and a new loss function that established the standard framework most commonly used today.

In this approach, knowledge transfer benefits from softening the output probability distribution of the teacher model. This is achieved by introducing a temperature parameter τ into the softmax function, which controls the smoothness of the distribution. This temperature-based Softmax is applied to both the teacher and the student during training, but it is set to 1 after the student has finished training.

$$q_i = \frac{\exp(z_i/\tau)}{\sum_j \exp(z_j/\tau)}$$

The key insight is that this added entropy provides the student with more information for each data point seen during training. When τ is high (e.g., $\tau > 1$), the resulting distribution becomes much more uniform, emphasizing the relative similarities between classes. Sometimes, this is referred to as "dark knowledge" contained within the network.

The Student model is trained to minimize a combined loss function that includes components corresponding to hard and soft targets. The total Knowledge Distillation Loss ($\mathcal{L}_{\text{KD}}$) is defined as a weighted sum of the Hard Loss ($\mathcal{L}_{\text{hard}}$) and the Soft Loss ($\mathcal{L}_{\text{soft}}$):

$$\mathcal{L}_{\text{KD}} = (1-\alpha) \cdot \mathcal{L}_{\text{hard}} + \alpha \cdot \mathcal{L}_{\text{soft}}$$

where

$$\mathcal{L}_{\text{hard}} = \mathcal{L}_{\text{CE}}(y_{\text{true}}, \sigma(\mathbf{z}_{\mathcal{S}}, 1))$$

$$\mathcal{L}_{\text{soft}} = \tau^2 \cdot \mathcal{L}_{\text{KL}}(\mathbf{P}^{\tau}, \mathbf{Q}^{\tau})$$

The soft loss, also known as the distillation loss, measures the difference between the Teacher's soft probabilities and the Student's soft probabilities using the Kullback-Leibler (KL) Divergence. The coefficient α balances the two loss terms.

Feature-based distillation goes beyond matching the final output probabilities and attempts to use information stored in the teacher's intermediate layers [3]. After applying a transformation to match the feature dimensions, the student learns to mimic the features of certain layers in the teacher. This is often achieved by minimizing the Mean Squared Error loss between the Student's and Teacher's intermediate layer activations. This process provides a powerful set of "hints" that guide the training of a deep student network. The feature matching loss is given by:

$$\mathcal{L}_{\text{Hintt}} = \frac{1}{2}||\mathbf{h}_{\mathcal{T}} - g(\mathbf{h}_{\mathcal{S}})||_2^2$$

where $\mathbf{h}_{\mathcal{T}}$ and $\mathbf{h}_{\mathcal{S}}$ are the feature maps of the Teacher and Student's corresponding hidden layers. Note that a function g is needed to match the dimensions of the two layers activations. This loss term is incorporated as an extra weighted loss term during the student's training.

B. Quantization

Quantization has become one of the most impactful ways to reduce the compute time and energy consumption of neural networks. The core principle is storing weight and activation tensors in lower bit precision than what they were trained in, by mapping continuous floating point values to a discrete set of quantized values. For example, moving from 32 bit to 8 bit representation uses 4x less memory for storage and 16x less compute for matrix multiplication. Many researchers have experimented over the years to demonstrate that model accuracy can be quite resilient to quantization, though it does introduce some noise that can reduce accuracy, especially at lower bit precision.

One of the first works related to quantization explored using second derivatives to measure weight importance and approximating the loss of removing the weight [9]. Although quantization eventually evolved to perturb weights rather than remove them, this paper was an important first step. In the 1990's, several papers such as [12] showed that neural networks could be trained using low precision arithmetic and still achieve high accuracy, which are perhaps some of the earliest implementations of quantization aware training. Researchers continued to build on these results and experiment with how to achieve high accuracy using other reduced formats such as dynamic fixed point and even using only binary weights constrained to 1 and -1 [10], [11]. Another important early work on quantization proposed a scheme using integer-only arithmetic [8]. They established core ideas such as the affine mapping of integers q to real numbers

$$r : r = S(q - Z)$$

for some constants S and Z representing the scale and zero point, respectively. These early contributions helped establish the two major paradigms used today: quantization aware training (QAT) and post training quantization (PTQ).

Post training quantization refers to techniques applied to pretrained models without requiring later retraining. It is the simpler of the two approaches as it requires less time and no additional training data, and still enables faster inference. A fundamental step in PTQ is finding good quantization ranges for each quantizer. The choice of range affects quantization error though a critical tradeoff between clipping thresholds and rounding error: to reduce clipping error, we can expand the quantization range by increasing the scale factor – but increasing the scale factor leads to increased rounding error. There are various techniques for navigating this tradeoff and choosing the range, including min-max (sensitive to outliers but simpler), mean squared error (less sensitive to outliers), and cross entropy (for the last layer), among others. In general PTQ works very well down to 8 bit quantization, but is not suitable for lower bit quantizations (4 bit and below for activations), where the accumulated quantization error becomes too large to recover without retraining.

Quantization aware training techniques simulate quantization during training on both weights and activations. By seeing where the noise sources are, the model is able to find more optimal solutions. QAT typically can give higher accuracy than PQT, but it comes with the added costs of training such as needing additional time and access to labeled training data. A challenge of PQT is that gradient calculations become complicated from the rounding operations, ending up as zero or undefined. To address this, we can approximate gradients using a straight through estimator [7]. QAT becomes necessary when targeting lower bit precision or when maximum accuracy is critical.

III. Methods

A. Workflow Overview

The model training and inference processes are managed through an automated pipeline that handles sequential input generation for the models and saves analysis outputs across different metrics. Performance tracking and profiling is handled by an internal tool called "smartengine prod," which is similar to other industry-standard tools like Weights and Biases. Experiments are managed via Pyplan configurations, which define various hyperparameters and model specifications for a given run in Python. Google uses Pyplan as a higher level abstraction layer on top of Python for assembling and configuring machine learning models. The models are trained on internal Google Cloud infrastructure using pufferlite TPUs. The model details themselves are proprietary to Google, and thus this project did not focus on model selection. Instead, we used the existing standard pConv architecture.

We optimized for both training and inference, and implemented our various changes around knowledge distillation and quantization as Pyplan configuration files. Many of these configurations also contain sweeps of a range of values for a given parameter. These configurations were then ported to Google's internal tooling, where experiments run on a copy of the models. Experiments are submitted to a scheduling queue and run on internal infrastructure, typically completing within a couple days. Results for each experiment are detailed in the smartengine prod metrics infrastructure. Experiments that have promising enough results can be promoted to run on live traffic.

B. Data

The dataset for the model features are roughly divided into query, user, and product features. Query features capture search intent and context, user features represent behavioral patterns and demographics, and product features contain information about the items being advertised. Due to privacy restrictions, our experiments focused on the model architecture and training methodology rather than making modifications to the feature side of things. The data about impressions are collected from Google search across browsers. The data about conversions are collected from Chrome and modeled for other browsers like Safari and DuckDuckGo. There is minimal preprocessing done on this dataset.

C. Experiments on Live Traffic

For live traffic evaluation, models are placed into serving cohorts where they handle real user queries on Google search, using their predictions to serve live traffic and advertisement placement and bidding decisions. This deployment process includes matching the brain server outputs from the model to those of the serving infrastructure, ensuring alignment across dimensions like predictions, uncertainty multipliers, and calibration factors. To qualify to be promoted to live traffic, a model needs to show improvements in two of the following three areas: regular metrics such as bias and loss, business metrics such as spend vs desired spend, and inference latency. These three areas were our main evaluation metrics.

Testing on live traffic is an important part of the development process because it can reveal other inefficiencies that might appear in the real world that did not arise in a controlled training environment. Our live experiments served 5% of live Google search traffic.

IV. Experiments

Our experiments compare our trained models against a comparable baseline that is used in production at Google. We chose to focus on this comparison because the teacher only predicts modeled conversions whereas total conversions are the primary metric of interest. Existing infrastructure also made extracting data for comparison challenging. By focusing on a comparable production model, we were also able to directly evaluate the potential business impact of our changes.

A. Knowledge Distillation

We implemented a base knowledge distillation setup using a teacher model to generate modeled (non-observed) conversions for the student model. The teacher model's pConvs predictions were combined with observed conversions and used as training labels for an auxiliary head in the student model. This was carried out by writing teacher predictions into a column which was later used by this auxiliary head during training. To improve learning for sparse, high-value conversions, we reweighted the distilled labels by a factor of the advertiser-set target cost per acquisition (tCPA). This reweighting was particularly important because Google's existing production student model struggles to learn high tCPA conversions effectively.

To optimize the knowledge distillation process, we conducted parameter sweeps over the temperature and alpha hyperparameters. The temperature parameter controls the softness of the probability distributions, while alpha balances the contribution of the distillation loss against the standard supervised loss.

B. Feature-Based Knowledge Distillation

In addition to output-level distillation, we explored feature-based knowledge distillation to transfer intermediate representations from the teacher to the student model. This set up also included a sweep over temperature and alpha values along with an additional weighting for the hint. The student architecture was identical to the other experiments. For simplicity, we added a single hint layer halfway through the student and teacher architectures. We used a linear projection and Mean Squared Error loss.

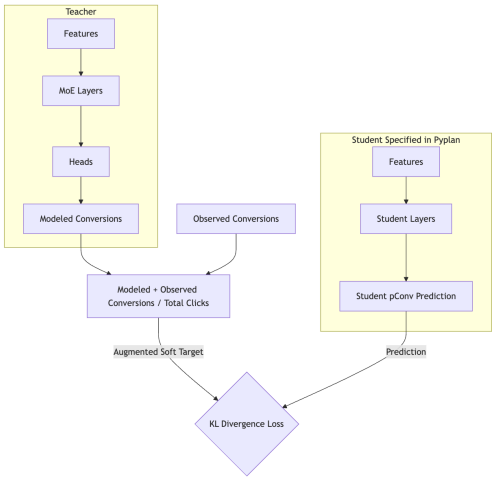

Fig. 2. Basic KD Setup

C. Quantization Combined with Knowledge Distillation

We applied quantization on top of the basic knowledge distillation setup to systematically evaluate the trade-off between model efficiency and prediction quality. Specifically, we focused on INT8 post-training quantization, as empirical analysis showed that reducing numerical precision yields substantial gains in inference latency and model size, while introducing only a limited degradation in predictive performance. This effect was particularly favorable when quantization was combined with knowledge distillation, which helped regularize the student model and improve robustness to low-precision representations.

Our final experimental configuration combined output-level knowledge distillation, feature-based knowledge distillation, and INT8 quantization, with additional tuning of temperature, loss weighting, and quantization parameters. This integrated approach allowed us to jointly optimize prediction quality and system efficiency, resulting in a compact, high-throughput student model suitable for large-scale production deployment.

V. Results

We observed substantial offline improvements with basic knowledge distillation, achieving a 0.70% reduction in PoissonLogLoss compared to the baseline production model. These gains were primarily driven by augmenting the total conversion label with the teacher's information about modeled conversions. Without distillation from the teacher, this information was not accessible in previous models. This change primarily improvement the model's accuracy, and there was no effect on latency since the model is still the same size as the baseline.

Implementing quantization on top of basic knowledge distillation reduced model accuracy as expected, yet it produced

gains in latency. The combined approach still slightly more accurate than the baseline production model, demonstrating that quantization could realize efficiency gains while maintaining some improvement in accuracy.

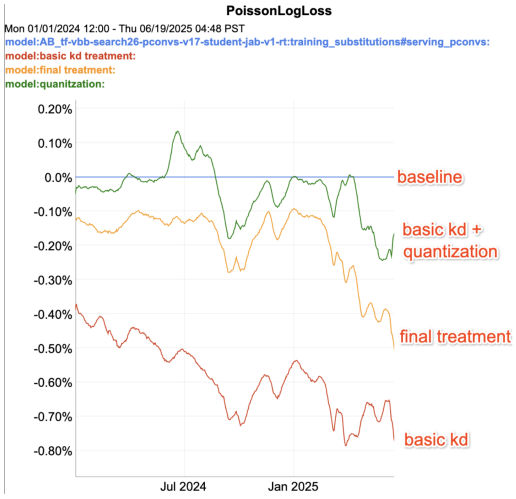

Fig. 3. PoissonLogLoss Across our Experiments

Our final candidate model, which combined basic knowledge distillation, quantization, and feature-based knowledge distillation with additional hyperparameter tuning, showed more consistent improvements compared to the baseline production model. Specifically, we achieved a 0.15% improvement in Poisson log loss. Since this model exhibited improvements in both loss and latency, we proceeded to test it on live traffic.

For Live Traffic Experimentation we added an arm for our model in the layer "Search29PconvsBiasTrack::No_Pds_Dpo_Copt" which contains all treatment models that serve live traffic and compared it with the existing production model control layer called "Search27pConvs::Control." The rows in our tables correspond to our control and treatment models, and Figure 4 is divided into total traffic and web traffic. In addition to latency, we evaluated the 99.9 percentile latency which reflects the latency of our slowest queries. This is crucial for measuring worst case scenarios which become significant as the amount of traffic scales. We observed a 0.14 ms improvement in 99.9% critical path latency, representing a meaningful reduction in the time required for the model to produce serving predictions.

The model changes also demonstrated positive return on investment for Google when tested on a slice of real world traffic. Figure 5 shows increases in the spend generated from our model with particularly strong performance in the high-value segment where target cost per action (tCPA) exceeded 250. When segmenting results by target CPA, we found that major quality improvements originated from the high target CPA slice of traffic. This slice had been previously underserved by the production model, confirming that our knowledge distillation approach with tCPA reweighting successfully addressed a critical gap in model performance for high-value conversions.

	Admixer 99.9%ile latency	Admixer critical path avg latency	Admixer 99%ile critical path latency	Admixer 99.9%ile critical path latency
Search27pConvs::Control 100312022 Details Links TOTAL	297.49	6.41	32.91	63.89
Search29PconvsBiasTrack::Arm_No_Pds_Dpo_Copt 103379085 Details Links TOTAL	297.50 0.01 [-0.01, 0.01] %	6.41 -0.00 [0.00, 0.00] ****	32.87 -0.03 [-0.11, -0.10] %	63.75 -0.14 [-0.25, -0.17] %
Search27pConvs::Control 100312022 Details Links WebSearch	310.53	5.47	34.80	94.26
Search29PconvsBiasTrack::Arm_No_Pds_Dpo_Copt 103379085 Details Links WebSearch	310.54 0.00 [-0.01, 0.01] %	5.46 -0.01 [-0.01, -0.01] ****	34.74 -0.06 [-0.18, -0.16] %	94.22 -0.04 [-0.06, -0.03] %

Fig. 4. Critical Path Latency Split by Total traffic and Websearch (only google.com)

	Spend	DesiredSpend	CappedDesiredSpend	ExpectedDesiredSpend	CappedExpectedDesiredSpend
Search29PconvsBiasTrack::Arm_No_Pds_Dpo_Copt 103379085 Details Links Reasonable * TARGET_CPA * TOTAL	0.10% [0.04, 0.16] % **○○	1.06% [0.72, 1.39] % ****	-0.75% [-0.91, -0.58] % ****	7.22% [7.02, 7.42] % ****	6.37% [6.30, 6.43] % ****
Search29PconvsBiasTrack::Arm_No_Pds_Dpo_Copt 103379085 Details Links Reasonable * TARGET_CPA * TCPA >250	19.45% [19.22, 19.69] % ****	9.53% [7.91, 11.16] % ****	2.49% [1.68, 3.30] % ****	34.03% [33.08, 34.99] % ****	30.66% [30.46, 30.87] % ****

Fig. 5. Spend and Desired Spend

VI. Discussion

A. Limitations

We were challenged early on to learn about and integrate with Google's infrastructure. Learning to use the Pyplan framework was also a challenge as we had to figure out how to translate standard PyTorch code into a declarative configuration file that their internal system would understand. This required thinking carefully about the various components and parameter settings, rather than simply writing step-by-step code. We also had limited access to documentation and examples for configuration files, and had to experiment a bit with what the system could and could not interpret. In general we were not able to get access to the internal systems for everyone and instead were dependent on our one team member for access to proprietary systems. Another constraint we faced was that we could not remove any model features or neurons since that logic is incompatible with Pyplans. While we wrote several experiments to explore pruning and quantization aware training, we were unable to integrate them due to these constraints. We also faced the constraint of a general production freeze at Google around the holidays, starting before Thanksgiving. This meant that we could not test some of our experiments on live traffic once the freeze went into place. All in all, these integration challenges meant that a substantial amount of project time was spent understanding the workflow and translating to configuration files rather than iterating through different optimization experiments.

B. Future Directions

Despite our promising results, the team at Google highlighted one concern that needed to be addressed before rolling out the model more broadly. Each prediction in their system is accompanied by a confidence interval which is used for validation, and if a prediction falls outside of that range, a

fall-back model must be used. This is known as a ”failed prediction.” Our student model required some lag in order to read predictions that are produced by the teacher. In practice, the lag associated with our KD implementation produced a small percentage of predictions that fell outside of this region and ”failed.” Even though this only happened in a small percentage of cases, we would like to implement some ideas from online knowledge distillation [4] to address these robustness concerns in the future.

VII. Conclusion

Our study demonstrates that meaningful efficiency gains in conversion prediction can be achieved without compromising predictive accuracy. By applying knowledge distillation and quantization, we were able to construct student models that satisfy stringent real-time serving constraints while incorporating the behavior of larger teacher architectures. Beyond numerical improvements, these results highlight a broader lesson: compression strategies, when carefully integrated into an existing production pipeline, can deliver system-level benefits that extend to user experience and advertising outcomes. Our findings reinforce the viability of compressed student models in high-throughput industrial systems and motivate continued exploration of efficient model design.

Acknowledgment

We would like to thank the Machine Learning Engineering team at Google for their feedback and support.

References

[1] C. Buciluǎ, R. Caruana, and A. Niculescu-Mizil, “Model compression,” in Proc. 12th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining (KDD), 2006, pp. 535–541.

[2] G. Hinton, O. Vinyals, and J. Dean, “Distilling the knowledge in a neural network,” arXiv preprint arXiv:1503.02531, 2015.

[3] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio, “FitNets: Hints for thin deep nets,” arXiv preprint arXiv:1412.6550, 2014.

[4] J. Gou, B. Yu, S. J. Maybank, and D. Tao, “Knowledge distillation: A survey,” International Journal of Computer Vision, vol. 129, pp. 1789–1819, 2021.

[5] R. Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper,” arXiv preprint arXiv:1806.08342, 2018.

[6] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, “A white paper on neural network quantization,” arXiv preprint arXiv:2106.08295, 2021.

[7] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” arXiv preprint arXiv:1308.3432, 2013.

[8] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2704–2713.

[9] Y. Le Cun, J. S. Denker, and S. A. Solla, “Optimal brain damage,” in Advances in Neural Information Processing Systems 2, D. S. Touretzky, Ed. San Mateo, CA: Morgan Kaufmann, 1990, pp. 598–605.

[10] M. Courbariaux, Y. Bengio, and J.-P. David, “Training deep neural networks with low precision multiplications,” arXiv preprint arXiv:1412.7024, 2015.

[11] M. Courbariaux, Y. Bengio, and J.-P. David, “BinaryConnect: Training deep neural networks with binary weights during propagations,” in Advances in Neural Information Processing Systems 28, 2015, pp. 3123–3131.

[12] M. A. Presley and M. R. Haggard, “A fixed point implementation of the backpropagation learning algorithm,” in Proc. IEEE SoutheastCon, 1994, pp. 89–93.