Secure Evaluation of Quantized Neural Networks

Assi Barak[*] Daniel Escudero[†] Anders Dalskov[†] Marcel Keller[‡]

Abstract

Machine Learning models, and specially convolutional neural networks (CNNs), are at the heart of many day-to-day applications like image classification and speech recognition. The need for evaluating such models whilst preserving the privacy of the input provided increases as the models are used for more information-sensitive tasks like DNA analysis or facial recognition. Research on evaluating CNNs securely has been very active during the last couple of years, e.g. Mohassel & Zhang (S&P'17) and Liu et al. (CCS'17), leading to very efficient frameworks like SecureNN (ePrint:2018:442), which can perform evaluation of some CNNs with a multplicative overhead of only 17–33 with respect to evaluation in the clear.

We contribute to this line of research by introducing a technique from the Machine Learning domain, namely *quantization*, which allows us to scale secure evaluation of CNNs to much larger networks without the accuracy loss that could happen by adapting the network to the MPC setting. Quantization is motivated by the deployment of ML models in resource-constrained devices, and we show it to be useful in the MPC setting as well. Our results show that it is possible to evaluate *realistic* models—specifically Google's MobileNets line of models for image recognition—within seconds.

Our performance gain can be mainly attributed to two key ingredients: One is the use of the three-party MPC protocol based on replicated secret sharing by Araki et al. (S&P'17), whose multiplication only requires sending one number per party. Moreover, it allows to evaluate arbitrary long dot products at the same communication cost of a single multiplication, which facilitates matrix multiplications considerably. The second main ingredient is the use of arithmetic modulo 2^{64}, for which we develop a set of primitives of indepedent interest that are necessary for the quantization like comparison and truncation by a secret shift.

1 Introduction

Machine Learning (ML) models are becoming more relevant in our day-to-day lives due to their ability to perform predictions on several types of data. Neural Networks (NNs), and in particular convolutional neural networks (CNNs), have emerged as a possible solution for many real-life problems such as facial recognition [36], image and video analysis for self-driving cars [7] and even for playing boardgames (most readers probably know of *AlphaGo* [52] which in 2016 beat one of the best Go players currently alive, Lee Sedol [60]). CNNs have also found applications within areas of medicine. [18], For example, demonstrates that CNNs are as effective as experts at detecting skin cancers from images.

In many applications the data on which the prediction is performed is sensitive and should ideally not be disclosed to the model owner. As an example, most airports now take

[*]Bar Ilan University
[†]Aarhus University
[‡]Data61

pictures of travelers faces when they pass through emigration and immigration control, and one could imagine that a facial recognition algorithm could be used to check for wanted terrorists, fugitives or *persona non grata* in general. However, this would require the airport to send images of all travelers to the entity performing the prediction—most likely a government agency—arguably violating peoples right to privacy. Similarly, it may be desirable to also keep the model hidden. For instance, the model might be the result of a hard training process performed by some company, and such an economical advantage should not be lost. It is nevertheless worth noting that an adversary with black-box access to a model, such as in the Machine Learning As A Service scenario, who can request predictions on arbitrary inputs can steal the model with near-perfect fidelity in some cases, as shown in [56, 58]. Defenses against such attacks exist (e.g. [44]) and can be seen as complimentary to this work.

In order to break this apparent contradiction (performing computation on data that is ought to be kept secret) tools like *secure multiparty computation* (MPC) can be used. Using these tools, the prediction can be performed so that it discloses neither the data nor the model. In the client-server model this is achieved by letting the data owner and the model owner *secret-share* their input towards a set of servers, who then run the computation over these shares.

Research in the area of secure evaluation of CNNs has been rich during the last couple of years, with very important works such as [22, 47, 49, 37, 40, 33, 57]. The main goal has been to reduce the performance gap between evaluating a CNN in the clear and doing it securely, and SecureNN [57], which is the state-of-the art in secure CNN evaluation, reports an overhead in the running time of a factor of 17–33 with respect to a non-secure evaluation in the clear. Even though this factor may be too high for some applications (such as the real-time image segmentation needed by self-driving cars), it may nevertheless be acceptable for many others, such as the example from earlier about detecting skin cancers in patients. However, we describe below observed patterns shared by some of these works which undermine their potential deployment in real-life scenarios.

No floating-point arithmetic and expensive activation functions. An important source of complexity that arises when securely evaluating a CNN using MPC is the fact that CNNs are based on floating-point arithmetic. This is problematic since secret-sharing-based MPC protocols are typically better suited for modular integer-based arithmetic. Several ways of overcoming this issue have been considered in the literature. One can map a set of floating-point numbers with finite precision to integers and use a modulus that is large enough so that overflows do not occur (effectively emulating integer arithmetic), as in [22]. However, this approach is prohibitive when many multiplications are to be performed. On the other hand, one can also use fixed-point arithmetic, i.e. a real number $x \in \mathbb{R}$ is approximated by $x \approx 2^{-\ell}\hat{x}$ where $\hat{x} \in \mathbb{Z}_M$ for some ℓ, M, where ℓ, the precision, is the same for all values. Under this representation, the arithmetic is performed over the integer representation $\hat{x}$. However, for this to work one must maintain a consistent representation for all the values. This is trivial to achieve for the addition of two fixed-point numbers, but, for a multiplication, the result must be truncated in order to reduce the precision and maintain the invariant. This can be done in MPC by computing the truncation using a binary garbled circuit or by executing a secret-sharing-based protocol for truncation. In either case, an additional overhead for supporting fixed-point arithmetic is introduced.

In the machine learning context there is an additional issue of using fixed-point arithmetic besides the complexity overhead, and it comes from the fact that reducing the precision may hurt the accuracy of the model's predictions. Well-tested models in the ML literature typically work with 32-bit or 64-bit floating-point arithmetic since it is very efficient when performed in the clear, but the effect on the accuracy of reducing the precision of the datatypes is not a trivial concern. Most previous work on secure ML treat this issue only superficially, by tweaking the precision parameters in an ad-hoc manner, usually dependent on the model under consideration. (An example of this is [8] in which the parameters for the homomorphic encryption scheme used depend on the model.) Moreover, the effect these heuristics have on the model's accuracy of prediction is not fully understood, in particular when considering model architectures beyond the one under consideration.

Finally, the complexity of some non-linear activation functions, like ReLU, often require the use of non-arithmetic MPC like garbled circuits, which imposes a significant overhead. This technique has been used, for instance, in [40, 37, 33], and it is reported in [57] that the most expensive part of the execution of these protocols lie in the garbled circuits. One way of avoiding the computation of such functions is to replace them by MPC-friendly alternatives that are cheaper to compute, as done, for instance, in SecureML [40].

Only small models are benchmarked. These works are mostly exclusively concerned with speed, and, understandably, they use smaller models (such as Cifar10 and MNIST) that are typically of little relevance in real-life applications. Moreover, as these models are usually significantly smaller than "real-life models", they provide little meaningful insight on how evaluation of larger and more complex models will fare, besides micro-benchmarks.

There are also little or no guarantees that the ad-hoc techniques for integer arithmetic mentioned before will scale well to more complex models, as these may have been developed and tested in the context of a specific benchmark model. Although it is true that in general many ML algorithms are based on empirical observations and heuristics, these are typically supported by an extensive and non-trivial research body from the ML domain, which deals with the choice of the architecture/parameters to obtain the best accuracy for a particular ML algorithm. We believe that this analysis should not be addressed superficially. Modifications to ML algorithms that are introduced when implementing them securely must be extensively studied to verify their validity. This includes techniques like the reduction in the precision or the modification of critical pieces like activation functions, which are necessary to boost the efficiency of secure ML algorithms.

Given the above we believe that secure solutions in the area of privacy-preserving ML should stick as much as possible to existing research and methods in the field of ML in order to guarantee, via a solid and extensive research body, the validity of the functionalities being implemented.

It is not clear how to extend these frameworks to other models. Even if these evaluation frameworks may extend easily to other CNNs in theory (ignoring the potential accuracy degradation we already mentioned), this is very hard to achieve in practice since the design of their implementations is not taking the user into account. As a consequence, even if the performance of certain CNN secure-evaluation framework may be acceptable for some applications, deploying it in a real-life scenario requires special care and expertise. This reduces the adoption of these techniques, and also the possibility of testing them in wider settings.

1.1 Our Contribution

It is not only the advances in MPC that may improve the viability of evaluating Machine Learning models in a secure way; recent research in the area of Machine Learning itself may be beneficial towards this goal. In this work we focus on developing solutions for securely evaluating a convolutional neural network (CNN) in an efficient manner, but we differ from previous works in that we address the issues mentioned above by relying on state-of-the-art machine learning research on the area of *quantization* instead of providing our own heuristic arguments for the validity of several ad-hoc modifications like reduced precision or replacing the activation functions (which may or may not work in more general scenarios).

Quantization, which is motivated by the deployment of ML models in resource-constrained environments like mobile phones or embedded devices, aims at reducing the size of neural networks by lowering the precision of the values involved. Furthermore, this also simplifies the arithmetic in some cases, leading to integer-only arithmetic and simpler activation functions. These techniques, quite coincidentally, are highly convenient when we consider a secure implementation using MPC.

By using quantization techniques we are able to tackle the issues mentioned in the previous section. We can summarize our main contributions as follow.

1. An extensive research body from the ML community supports the validity and amplitude of our techniques, thus it is clear that our protocol extends correctly to other settings without undermining accuracy substantially. We focus on the quantization scheme developed by Jacob et al. [32], which is already implemented in Tensorflow Lite and has been demonstrated to work outstandingly well in several challenging prediction tasks.

2. Secondly, we benchmark our protocol using models that are of practical importance and already in wide use. In particular, we use models from the MobileNets architecture [28] which have demonstrated very good accuracy (almost 90% top-5 accuracy on the LSVRC dataset for the largest V2 model) despite their small size (between 500kb and 4.3mb depending on choice of hyperparameters).

3. In the process of evaluating the models above we implement in our protocol some of the basic kernels for CNNs included in the Tensorflow Lite framework like convolutions, depthwise convolutions and more. This ultimately means that, in addition to supporting secure evaluation of MobileNets networks (which are already useful for a very broad range of applications), a large family of TFLite models are supported, which we believe is an important milestone towards deploying secure ML in real-life scenarios.

4. Finally, we show through benchmarks that *all* of the currently available MobileNets models are feasible to compute within reasonable time and communication using our protocol (between 1 and 11 seconds, and 1.2 and 15 gb, respectively for the semi-honest case). Moreover, we demonstrate this fact both for a semi-honest protocol over $\mathbb{Z}_{2^{64}}$, *and* for an *actively* secure protocol over a prime field. To the best of our knowledge, this is first result showing that inference in MPC with active security is possible.

In this work we devise the first consistent, scalable method for private evaluation of machine learning models, which can be applied accurately and efficiently to real-world CNNs with millions of weights by combining recent research in deep learning (quantization) and

MPC optimization. We demonstrate evaluating some of the smaller MobileNetsV1 models in less than 2 seconds. Our approach is easily extensible to additional security models and computation domains using the SPDZ multi-protocol compiler, and is within engineering range to a full-blown private model evaluation framework. Moreover, it is important to add that TFLite already provides methods to take any (floating-point) Tensorflow model and convert it into a quantized model with only a small loss of precision. All these tools together provide a complete pipeline to execute virtually *any* neural network that is trained in Tensorflow.[1] Given that Tensorflow is one of the most important industrial-grade frameworks for machine learning, we expect our work to be a key step towards the widespread adoption of MPC techniques to enhance privacy in current services and platforms.

Our protocol constitutes the first MPC-based solution to privacy-preserving CNN evaluation using quantization. It is developed in the client-server model, in which a set of clients secret-share the model and the data towards a set of servers, which then execute the protocol. We use three servers, and the data and the model remain secret even if at most one server is passively corrupted (semi-honest security).

Since our work is the first in using quantization for secure evaluation of CNNs in MPC, and since we use different NN architectures than previous work, establishing a fair comparison with other work is not a trivial task. We do however extrapolate from our micro-benchmarks in Section 4.2 a rough estimate of the performance of our protocol on the networks considered in SecureNN [57], which is the state of the art in secure neural network evaluation and whose networks are the same as those used in SecureML [40], Chameleon [47] and MiniONN [37]. Our result show that our protocol is in general faster, but that it sends more data for models that contain more non-convolution layers.

1.2 Techniques

1.2.1 Quantization

As mentioned in the preceding section, the core of our work is the theory of quantization. In a nutshell, this allows a set of real numbers $\{\alpha_1, \dots, \alpha_n\} \in \mathbb{R}$ to be represented by a set of integers $\{a_1, \dots, a_n\} \in \mathbb{Z}_M$ in a way so that basic operations such as additions and multiplications are preserved, at least up to some extent. In the context of CNNs the motivation of using quantization is to reduce storage: Instead of storing a set of real numbers (say, 32-bit floating-point numbers), one can replace this set by the corresponding integers; in practice either 8 or 16-bits.

Research in developing quantization schemes for several ML models and understanding the effect of these techniques in the accuracy is extensive in the ML community, as can be seen from the very recent survey by Guo [25]. We choose to focus our work on the quantization scheme by Jacob et al. [32], which works by mapping a real number $\alpha_i \in \mathbb{R}$ to $x_i \in \mathbb{Z}_M$ such that $\alpha_i \approx m \cdot (x_i - z)$, where $m \in \mathbb{R}$ and $z \in \mathbb{Z}_M$ are parameters depending only on the set being quantized and the bound M. This affine mapping, which can be seen as a shifted version of fixed-point arithmetic, turns out to preserve accuracy quite well even when using 8-bit integers [34].

By making use of this quantization scheme we can implement the core operations used in CNNs, like convolutions and fully connected layers, by using mostly integer-only arithmetic. Because these operations essentially rely on taking dot products, which can be done very easily with this quantization scheme, by taking the corresponding integer dot product over the quantized values and then performing a truncation afterwards. However, as we will see, the two approaches differ in a number of places, especially when considering

[1]We remark that we do not support currently all the kernels available in TFLite.

activation functions and special layers like batch normalization. Moreover, the effect of using this type of quantization in the accuracy of the model has been studied extensively already and has been found to be very small. For example, some of the MobileNets models we consider in this work still achieve above 70% success rate on the ImageNet dataset.

1.2.2 Secure Computation

We implement our CNN interpreter using the ring version of the 3-party replicated-secret-sharing-based protocol from [3], which is secure against one passive corruption. The ring $\mathbb{Z}_{2^{64}}$ fits naturally with the quantization scheme, as described in [32], and many bit-operations over this ring can be done much more efficiently than its field counterpart, as shown in [2]. We use MP-SPDZ [41] because it provides an efficient implementation of most of the necessary primitives as well as high-level programming interface for it, and we added missing routines like truncation by a secret shift, as shown in Section 3.2.2, and cheap sums-of-products, which are at the core of our efficiency gains. Furthermore, the use of the MP-SPDZ compiler allows us to write our CNN evaluator in a clean, Python-like programming language, which is easy to extend to other models.

We outline our implementation of various basic algorithms such as comparison and truncation in the context of arithmetic circuits over $\mathbb{Z}_{2^{64}}$. There is a body of work for the case of fields of prime order [9, 10], but to the best of our knowledge, there is no prior work for rings such as the one used here.

Finally, as we already hinted above, a significant advantage of protocols based on multiplicative secret sharing such as ours consists in allowing for cheap computation of dot products, which is an essential operation in CNNs. This protocol, just like any other secret-sharing based protocol, requires interaction to process any multiplication. However, due to the special properties of multiplicative secret sharing, the dot product of two vectors of *any* length can be computed at the communication cost of only one single multiplication, which is one ring element per party, as shown in Section 3.1. This optimization is at the core of our performance improvement with respect to previous work.

1.3 Related Work

Early work in the area of evaluating neural networks securely can be traced back at least to the work by Orlandi et al. [5, 43], which is mostly based on HE techniques. CryptoNets [22] uses Leveled Homomorphic Encryption (LHE). However, due to the limitations on this cryptographic primitive, several ad-hoc alternatives to the activation functions are introduced, as well as alternatives to some other layers in the CNN like max pooling. As we have already argued, the validity of introducing such methods in terms of the accuracy obtained is questionable. Moreover, for the case of CryptoNets this is explicitly addressed in [11] where the ad-hoc activation function (squaring) is replaced by a polynomial approximation of known activations like ReLU. Unfortunately, the degree of these polynomials needed to provide an acceptable accuracy is prohibitive for LHE. Lastly, CryptoNets only performs evaluation of MNIST.

SecureML [40] studies the problem of securely training and evaluating different ML models like linear/logistic regression and convolutional networks. Their protocol achieves passive security in the two-server model, and it is based on additive secret sharing over a large enough ring. Non-linear activation functions (which are expensive on arithmetic shares) are handled by switching between arithmetic secret sharing and using a garbled circuit instead. However, to this end SecureML introduces new ad-hoc activation functions that can be represented by suitable binary circuits. More precisely, SecureML uses fixed-point arithmetic with a simple truncation mechanism which works thanks to the particular 2-party scenario with additive secret sharing: Each party truncates its share locally. However, for this to work without a significant error probability, either the ring has to be large or the number of multiplications between each truncation have to be small. How to determine these parameters will depend on the concrete model being computed and it is not obvious how to select them.

This approach yields an improvement of a factor of 4–8 with respect to previous solutions to the arithmetic issue like embedding the computations in a large enough field to avoid overflows (as done in Cryptonets [22]) or using a garbled circuit to perform fixed-point [21] or floating-point [42] multiplication. However, this truncation mechanism is not perfect and it has an error of ± 1 with some probability, which can be made arbitrarily small by choosing a ring/field that is much larger than the one needed for the actual computation (see Theorem 1 in [40]). Although the techniques sketched above to fit neural network training/evaluation to the MPC setting (alternative activation functions and probabilistic fixed-point arithmetic) improve efficiency, their effect in the accuracy is not completely understood. Mohassel et al. [40] provide arguments and intuition on this matter, but they only perform tests for very restricted datasets: MNIST for neural networks and Arcene for regression.

MiniONN [37] is the first work that identifies and tackles the problems we have highlighted on SecureML and CryptoNets about the modifications on existing neural network models and builds on top of [40] to address these issues. The authors aim at providing a framework that implements the most basic operations involved in common NNs like linear transformations and activation functions. This work uses polynomial splines to approximate widely-used nonlinear functions in ML like sigmoid and tanh, which yields a negligible loss in accuracy. Arithmetic is handled like in [21] by using a garbled circuit to scale down the data. More precisely, the authors scale the floating-point numbers up to integers using the same factor for all values and then drop the fractional parts using a garbled circuit. The effect of this approach, which can be already seen as some form of quantization, has not been extensively tested, not even in the reference [39] provided by the authors on this matter. MiniONN works in the 2-party setting by using a mix of secure 2-party computation (namely, garbled circuits) and homomorphic encryption.

Chameleon [47] achieves faster running times due to a technique that uses an additional party that acts as a helper in order to avoid expensive protocols like oblivious transfer, which is used for example in SecureML. Chameleon uses conversions between garbled circuits and additive shares, and it is based on the ABY framework [17]. However, despite the performance improvement of Chameleon with respect to previous work, this work is only benchmarked using fixed-point arithmetic on small models like MNIST, and it is not clear what the impact in accuracy of this type of finite-precision arithmetic for a larger model would be. Furthermore, the authors left as future work extending their techniques to the floating-point setting like those from [1], which would achieve more fidelity with respect to the evaluation of a model in the clear.

Gazelle [33] uses a hybrid approach that combines additive homomorphic encryption with garbled circuits. Their benchmarking is done using Cifar10 and MNIST datasets, and they achieve an improvement of a factor between 20 and 30 with respect to MiniONN and Chameleon, respectively. However, like Chameleon, Gazelle uses fixed-point arithmetic and the selection of parameters like the amount of precision is done in an ad-hoc per-model basis whose generality is not fully understood.

DeepSecure [49] focuses on using garbled circuits (GC) to implement a library containing several basic building blocks for evaluating neural networks like two-dimensional convolutions, pooling (both max and mean), fully connected and non-linearities like softmax, sigmoid, Tanh and ReLU. This work focuses on performance, introducing several optimizations on the GC protocol to improve the speed of the inference. The authors benchmark their implementation using several networks for visual and audio recognition and smart-sensing, and the errors due to fixed-point arithmetic and approximation of activations are analyzed in a per-operation basis. It is important to remark that DeepSecure's library supports floating-point arithmetic as well.

Finally, SecureNN [57], which is the state-of-the-art protocol in secure training and inference of CNNs, is one of the few solutions which is solely based on additive secret sharing only. The protocol is information-theoretically secure and it is developed for the three and four-party settings. Part of the performance gain comes from a technique that allows the parties to securely compute integer comparisons cheaply, which is as key ingredient for some activation functions such as ReLU. However, this method only works for odd-modulus arithmetic while for efficiency reasons SecureNN uses arithmetic modulo powers of 2. To allow for the best of both worlds, the authors also introduce a simple but effective method to move from one sharing to the other. Additionally, their sharing scheme is not symmetric for all parties, i.e. one of the parties has a different role in the computation, which simplifies many of the expensive procedures like generating random multiplication triples [6]. Despite the improved efficiency of SecureNN, the impact in accuracy of using fixed-point arithmetic is not discussed by the authors.

Quantization in prior work. Whether implicitly or explicitly, most prior work uses some form of quantization already. For instance, replacing directly floating-point by fixed-point numbers can already be seen as quantization. However, as we have already argued extensively, more often than not this is done in a not very robust manner. Relatively little work has made explicit use of quantization in the context of securely evaluating Machine Learning models. One example is the recent work by Bourse et al. [8], where the authors use a quantization technique that is similar to the one described in the by Courbariaux and Bengio [12]. Another recent work which use the same techniques is [51]. Nevertheless, their work lies in the FHE domain, which differs from multiparty computation. For instance, the fact that the weights are kept in the clear by the model owner changes the way the computation is performed, and allows them to use only additions and subtractions.

1.4 Outline of the Document

In Section 2 we give a brief introduction to neural networks and then we describe quantization in this domain, focusing on the quantization scheme by Jacob et al. [32]. Then in Section 3 we provide a self-contained description of our protocol for secure inference, describing the basic building blocks. We discuss implementation details and present benchmarks in Section 4, and conclude in Section 5.

2 Quantization

Deep learning models are at the core of many real-world tasks like computer vision, natural language processing and speech recognition. However, in spite of their high accuracy for many such tasks, their usage on embedded devices like mobile phones, which have tight resource constrains, becomes restricted by the large amount of storage required to store the model and the high amount of energy consumption when carrying the computations, which are typically done over floating-point numbers. To this end, researchers in the machine learning community have developed techniques that allow weights to be represented by low-width integers instead of the usual 32-bit floating-point numbers, and quantization is recognized to be the most effective such technique when the storage/accuracy ratio is taken into account.

Quantization allows the representation of the weights and activations to be as low as 8 bits, or even 1 bit in some cases [12, 46].[2] This is a long-standing research area, with initial works already dating back to the 1990s (e.g. [19, 4, 55, 38]), and this extensive research body have enabled modern quantized neural networks to have essentially the same accuracy as their full-precision counterparts [13, 62, 24, 26, 45], even with very large CNN architectures like AlexNet [35], VGGNet [53], GoogleNet [54] and ResNet [27].

It is not our goal to provide a survey on quantization techniques, and we refer the reader to Guo [25] for a thorough discussion on the topic. However, we do want to illustrate that quantization is an effective mechanism to reduce the size of a CNN and simplify computation without affecting accuracy significantly, and that this is due to a very large, long-standing, non-trivial and active research body.

In what follows we provide a superficial but sufficient introduction to Deep learning, and then we present the quantization scheme our work is based on Jacob et al. [32], which is already implemented in Tensorflow.

2.1 Notation

For a value $\boldsymbol{x} \in \mathbb{R}^{N_1 \times N_2 \times N_3}$ we use $\boldsymbol{x}[i, j, c] \in \mathbb{R}$ to denote taking i'th value across the first dimension, the j'th value across the second dimension and the c'th value across the last dimension. In a similar way, we might write $\boldsymbol{x}[\cdot, \cdot, c] \in \mathbb{R}^{N_1 \times N_2}$ to denote the matrix obtained by fixing a specific value for the last dimension. A real value interval is denoted by $[a, b]$ and a discrete interval by $[a, b]_{\mathbb{Z}}$. We define *clamping* of a value $x \in \mathbb{R}$ to the interval $[a, b]$, denoted by $\mathsf{clamp}_{a,b}(x)$, by setting $x \leftarrow a$ if $x < a$, $x \leftarrow b$ if $x > b$ and otherwise $x \leftarrow x$. (Clamping to a discrete interval is similarly defined.) We denote by $\mathbb{N}_\ell$ the set $\{1, \ldots, \ell\}$.

2.2 Deep Learning

An artificial neural network, or simply neural network (NN) for short, is a machine learning model that is used to obtain predictions on some data that has proven to be very successful at specific tasks like character recognition, data processing, classification, and many more. In a very general setting, a neural network is an ordered set of functions $(f_1, \ldots, f_n)$ where $f_i : D_{i-1} \to D_i$, with D_i some space of the form $\mathbb{R}^{N_1 \times \cdots \times N_{\ell_i}}$. An element of such a set is known as a *tensor*, and each function f_i is known as a *layer*. The input to the neural network is a tensor $\boldsymbol{x} \in D_0$, and the output is $\boldsymbol{y} = f_n \circ \cdots \circ f_1(\boldsymbol{x}) \in D_n$. For convolutional neural networks (CNNs) in practice D_0 could be $\mathbb{R}^{128 \times 128 \times 3}$ to represent 128×128 images with 3 color channels (RGB), and the output set D_n could be a vector where the i-th entry represents the probability that the input image has a given label indexed by i. See Figure 1 for a visual representation of a neural network.

Some of the layers considered typically in practice include affine layers like fully connected and convolutional layers, non-linear activations like ReLU and ReLU6, and downsampling layers like average or max pooling. Below we discuss some of the layers we will consider in this work.

[2]Furthermore, some quantization techniques also allow to represent gradients with a small number of bits, which effectively allows for quantized training of neural networks. However, this is still in a very early stage, and since we are focused only on inference in this work, we do not present such techniques.

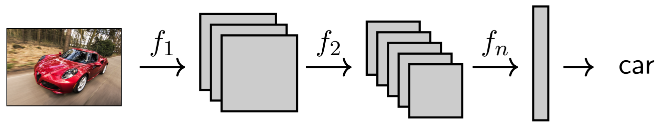


Figure 1: Visualization of a Convolutional Neural Network.

Two-Dimensional Convolutional Layer. A two-dimensional convolution is an operation performed between two three-dimensional tensors to produce another three-dimensional tensor. The parameters for a two-dimensional convolution are the input dimensions I_h, I_w, I_d, the output dimensions O_h, O_w, O_d and the dimension of the windows W_h, W_w. On input $\boldsymbol{x} \in \mathbb{R}^{I_h \times I_w \times I_d}$, a weight tensor $\boldsymbol{w} \in \mathbb{R}^{O_d \times W_h \times W_w \times I_d}$ and a bias vector $\boldsymbol{b} \in \mathbb{R}^{O_d}$, the convolution performs the following operations.

1. For each $(i, j) \in \mathbb{N}_{O_h} \times \mathbb{N}_{O_w}$, a sub-tensor of $\boldsymbol{x}$, denoted by $\boldsymbol{x}_{i,j} \in \mathbb{R}^{W_h \times W_w \times I_d}$, is extracted. We will not dive into the details of how this is done besides saying that it depends on some extra parameters like the *stride* and the *padding type*, which are part of the architecture.[3]

2. The entry indexed by $(i, j, k) \in \mathbb{N}_{O_h} \times \mathbb{N}_{O_w} \times \mathbb{N}_{O_d}$ of the output is computed as $\mathsf{dot}(\boldsymbol{x}_{i.j}\boldsymbol{w}[k, \cdot, \cdot, \cdot]) + \boldsymbol{b}[k]$, where dot represents the sum of the point-wise products of the two inputs.

ReLU and ReLU6. It should be clear from the description of the convolutional layer above that this operation is affine, and any composition of such functions would result in an affine function as well. The purpose of non-linear activation functions like ReLU and ReLU6, as the name suggests, is to break this linear relation between input and output by using non-linear operations. ReLU, which stands for rectified linear unit, takes as input a tensor and applies the following to each one of its entries: if the entry is negative then replace it with 0, and leave it unchanged otherwise. ReLU6 works in a similar way, but if the entry is greater than 6 then it is replaced by 6. This can be seen as clamping the input to the interval $[0, 6]$.

Average and Max Pooling. A pooling operation takes as input a tensor $\boldsymbol{x} \in \mathbb{R}^{I_h \times I_w \times I_o}$ and returns a tensor $\boldsymbol{y} \in \mathbb{R}^{O_h \times O_w \times O_d}$ with $O_h < I_h$, $O_w < I_w$ and $O_d = I_d$, which can be seen as a *down-sampling* operation. Each entry $\boldsymbol{y}[i, j, k]$ is computed by applying a given operation to the entries of some sub-matrix $\boldsymbol{x}_{i,j}[\cdot, \cdot, k]$ of $\boldsymbol{x}[\cdot, \cdot, k]$, which is extracted in a similar way as in a convolution. For average pooling the operation is just the mean of the entries in the matrix, and for max pooling the maximum of the entries is returned.

Output. The output of a CNN is typically a vector. The index with the maximum value in this vector represents the most likely label corresponding to the input, and this vector can be normalized into a probability distribution via a monotonous function like Softmax so that the value corresponding to index i is the probability that the input has label i.

2.3 Google's Quantization Scheme

Now that we have seen some of the different operations involved in a typical CNN, we describe one possible way of quantizing these operations. This is based on the quantization

[3]See `https://cs231n.github.io/convolutional-networks/#conv` for an elementary and thorough explanation.

scheme developed by Google's researchers [32, 34], which is part of Tensorflow Lite,[4] a library for running machine learning models on mobile and embedded devices. We choose this quantization scheme for the following reasons.

Simplicity. The procedures for quantization and de-quantization are very simple and suitable for our MPC protocol.

Reference Implementation. Since their scheme is already implemented in TFLite, this allows us to contrast our implementation with theirs to make sure we match their models with as high fidelity as possible.

Framework for Testing. TFLite already provides us with some pre-trained quantized models that facilitate the testing. Moreover, as we will see in Section 4, it also provides a fully functional pipeline to convert *any* floating-point model into a quantized model with little to none loss in accuracy, which extends the impact of our results.

High Impact. Many widely-used products of Google and its sister companies are already using TFLite for their prediction tasks. We believe that integrating secure computation into these platforms allow for a more rapid deployment of these technologies.

2.3.1 Quantization and De-Quantization

The scheme comes in two variants, one for 8-bit integers and another one for 16-bit integers. In this work we focus in the former, and we provide our description only in that setting.

Let $m \in \mathbb{R}$ and $z \in [0, 2^8)_{\mathbb{Z}}$ and consider the function $\mathsf{dequant}_{m,z} : [0, 2^8)_{\mathbb{Z}} \to \mathbb{R}$ given by $\mathsf{dequant}_{m,z}(x) = m \cdot (x - z)$. This function transforms the interval $[0, 2^8)_{\mathbb{Z}}$ injectively into the interval $I = [-m \cdot z, m \cdot (2^8 - 1 - z))$ and as such it admits and inverse $\mathsf{quant}_{m,z}$ mapping elements in the image of $\mathsf{dequant}_{m.z}$ into $[0, 2^8)_{\mathbb{Z}}$. We define the quantization of a number $\alpha \in I$ to be $\mathsf{quant}_{m,z}(\alpha')$, where α' is closest number to α such that α' is in the image of $\mathsf{dequant}_{m,z}$.

The constants m, z above are the parameters of the quantization, and are known as the *scale* and the *zero-point*, respectively. This quantization method will be applied on a per-tensor basis, i.e. each individual tensor $\boldsymbol{\alpha}$ has a single pair m, z associated to it. These parameters are determined at training time by recording the ranges on which the entries of a given tensor lie, and computing m, z such that the interval $[-m \cdot z, m \cdot (2^8 - 1 - z))$ is large enough to hold these values. See Figure 2 for a visualization of this quantization method, and see Jacob et al. [32] for details.

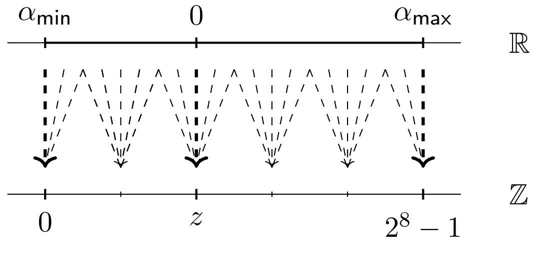


Figure 2: Visualization of the Quantization Scheme in [32]. The continuous interval on top is mapped to the discrete interval below, and multiple numbers may map to the same integer due to the rounding.

[4]https://www.tensorflow.org/lite/

2.3.2 Dot Products

Computing dot products is a core arithmetic operation in any CNN. We discuss this how to do this with the quantization method described above.

Let $\boldsymbol{\alpha} = (\alpha_1, \dots, \alpha_N)$ and $\boldsymbol{\beta} = (\beta_1, \dots, \beta_N)$ be two vectors of numbers with quantization parameters (m_1, z_1) and (m_2, z_2), respectively. Let $\gamma = \sum_{i=1}^{N} \alpha_i \cdot \beta_i$, and suppose that γ is part of a tensor whose quantization parameters are (m_3, z_3). Let $c = \mathsf{quant}_{m_3,z_3}(\gamma)$, $a_i = \mathsf{quant}_{m_1,z_1}(\alpha_i)$ and $b_i = \mathsf{quant}_{m_2,z_2}(\beta_i)$. It turns out we can compute c from all the a_i, b_i by using integer-only arithmetic and fixed-point multiplication, as shown in the following.

Since $\gamma \approx m_3 \cdot (c - z_3)$, $\alpha_i \approx m_1(a_i - z_1)$ and $\beta_i \approx m_2(b_i - z_2)$, it holds that

$$m_3 \cdot (c - z_3) \approx \gamma = \sum_{i=1}^{N} \alpha_i \cdot \beta_i \approx \sum_{i=1}^{N} m_1 \cdot (a_i - z_1) \cdot m_2 \cdot (b_i - z_3).$$

Hence, we can approximate c as

$$c = z_3 + \frac{m_1 \cdot m_2}{m_3} \cdot \sum_{i=1}^{N} (a_i - z_1) \cdot (b_i - z_2) \tag{1}$$

The summation $s = \sum_{i=1}^{N} (a_i - z_1) \cdot (b_i - z_2)$ involves integer-only arithmetic and it is guaranteed to fit in $16 \log N$ bits, since each summand, being the product of two 8-bit integers, fits in 16 bits. However, since $m = (m_1 m_2)/m_3$ is a float, the product $m \cdot s$ cannot be done with integer-only arithmetic. This product is handled in TFLite by essentially transforming m into a fixed-point number and then performing fixed-point multiplication, rounding to the nearest integer. More precisely, m is first normalized as $m = 2^{-n} m''$ where $m'' \in [0.5, 1)$,[5] and then m'' is approximated as $m'' \approx 2^{-31} m'$, where m' is a 32-bit integer. This is highly accurate since $m'' \geq 1/2$, so there are at least 30 bits of relative accuracy.

Thus, given the above, the multiplication $m \cdot s$ is done by computing the integer product $m \cdot s$, which fits in 64 bits since both m and s use at most 32 bits (if $N \leq 2^{16}$), and then multiplying by 2^{-n-31} followed by a rounding-to-nearest operation. Finally, addition with z_3 is done as simple integer addition.

If the quantization parameters for γ were computed correctly, it should be the case, by construction, that the result c lies in the correct interval $[0, 2^8)_{\mathbb{Z}}$. However, due to the different rounding errors that can occur above, this may not be the case. Thus, the result obtained with the previous steps is clamped into the interval $[0, 2^8)_{\mathbb{Z}}$.

Addition of bias. In the context of CNNs the dot products above will come from two-dimensional convolutions. However, these operations not only involve dot products but also the addition of a single number, the bias. In order to handle this in a smooth manner with respect to the dot product above, the scale for the bias is set as $m_1 m_2 / m_3$ and the zero-point it set to 0. This allows the quantized bias to be placed inside the summation s, involving no further changes to our description above.

[5] Jacob et al. [32] find that in practice $m \in [0, 1)$, which is the reason why such normalization is possible. We also confirm this observation in our experiments, although it is not hard to extend this to the general case (in fact, TFLite already supports it).

2.3.3 Other layers.

Other layers like ReLU, ReLU6 or max pooling, which involve only comparisons, can be implemented with relative ease directly on the quantized values, assuming these share the same quantization parameters. This is because if $\alpha = m(a - z)$ and $\beta = m(b - z)$, then $\alpha \leq \beta$ if and only if $a \leq b$, so the comparisons can be performed directly on the quantized values.

In fact, activations like ReLU6 (which is used extensively in the models we consider in this work) can be entirely fused into the dot product that precedes it, as shown in Section 2.4 of [32]. Since ReLU6 is essentially a clamping operation, it is possible, by carefully picking the quantization parameters, to make the clamping of the product to the interval $[0, 2^8)_{\mathbb{Z}}$ *also* take care of the ReLU6 operation. In short, if the zero-point is 0 and the scale is 6/255, then we are guaranteed that $m(q - z) \in [0, 6]$ for any $q \in \{0, \dots, 2^8 - 1\}$.

On the other hand, mathematical functions like sigmoid must be handled differently. We will not be concerned with this type of functions in this document since it is the case in practice that ReLU and ReLU6 (or similar activation functions) are typically enough.[6]

3 Secure Evaluation of Quantized Neural Networks

Here we describe our protocol for secure evaluation of quantized neural networks. We begin by describing the MPC protocol our work is based on in Section 3.1, and then we proceed to describe the different components of the NN evaluation.

3.1 Replicated-SS-based MPC Over Rings

Consider a setting with three parties P_1, P_2, P_3 among which at most one is corrupted by a passive adversary. We use the protocol by Araki et al. [3], which is based on replicated secret sharing, and we primarily employ it in the ring defined by arithmetic modulo 2^{64}. In said protocol, a value $x \in \mathbb{Z}_{2^{64}}$ is secret-shared among P_1, P_2, P_3 if each P_i holds a random pair (x_i, x_{i-1}) (indexes wrap modulo 3) such that $x_1 + x_2 + x_3 = x \bmod 2^{64}$. We denote this sharing by $\langle x \rangle$. Clearly, if the adversary only controls one party passively, it cannot learn anything about x.

It is easy to see that addition of two shared values can be done locally by letting each party add its shares component-wise, and so can multiplication by a public value. However, multiplication of two shared values requires interaction. To this end, it is assumed that the parties hold additive shares of zero, i.e. each party P_i holds $\alpha_i \in \mathbb{Z}_{2^{64}}$ such that $\alpha_1 + \alpha_2 + \alpha_3 = 0$. Then, in order to multiply $\langle x \rangle$ and $\langle y \rangle$, each P_i computes $t_i = x_i y_i + x_i y_{i_1} + x_{i_1} y_i + \alpha_i$ and sends this value to P_{i+1}. Then P_i defines its share of $x \cdot y$ to be the pair (t_i, t_{i-1}), where t_{i-1} was received from party P_{i-1}.

This protocol involves sending only one ring element per party to compute shares of a product. Moreover, the sharing of zero above can be preprocessed in a very efficient manner by an initial set-up phase in which each P_i sends a key to P_{i+1} for a pseudorandom function [15, 20].

Efficient Sums-Of-Products. Suppose that the parties have shares $\left\langle x^1 \right\rangle, \dots \langle x^n \rangle$ and $\left\langle y^1 \right\rangle, \dots, \langle y^n \rangle$, and they want to compute shares of the dot product, or sum-of-product $z = \sum_{i=1}^{n} x^i \cdot y^i$. A naive way of doing it is to compute shares of each product $x^i \cdot y^i$ as we have shown above and then let each party add its shares locally. However, applying this

[6]See [32] for a discussion on quantization of mathematical functions.

method would have the communication cost of n individual multiplication, which means n ring elements per party. Instead, a simple observation allows this dot product to be computed at a communication cost which is completely independent of n. This turns out to be a key optimization tool for our particular application since most of the computations in a CNN are precisely dot products.

In order to obtain shares of z, each party P_i computes $t_i = \sum_{j=1}^{n} \left(x_i^j y_i^j + x_i^j y_{i_1}^j + x_{i_1}^j y_i^j \right) + \alpha_i$, where α_i is defined as before, and P_i sends this value to P_{i+1}. Then P_{i+1} defines its share of z to be (t_i, t_{i-1}). A straightforward calculation shows that this produces shares of z, and it was done at the communication cost of one single multiplication.

3.2 Quantized CNNs in MPC

Now we turn to the question of how to implement inference of a quantized CNN, using the quantization scheme we described in Section 2.3 using the MPC protocol from the previous section. We begin by describing the setting.

Recall from Section 2 that each weight tensor $\boldsymbol{a}$ in a quantized CNN has a scale $m \in \mathbb{R}$ and a zero-point $z \in \mathbb{Z}_{2^8}$ associated to it, such that $\alpha \approx m \cdot (a - z)$ is the actual floating-point numbers corresponding to each 8-bit integer a in the tensor. Also, biases are quantized in a similar manner but with a 32-bit integer instead, a zero point of zero, and a scale that depends on the inputs and output to the layer it belongs to, as explained in Section 2.3.2. We assume that the model owner, who knows all this information, distributes to the three parties P_1, P_2, P_3 shares using the scheme described above of the quantized weights and biases of each layer in the network.[7] Also, the zero points associated to each tensor is shared towards the partes.

The scales of the model, on the other hand, are handled in a slightly different way. Each dot product in the quantized network requires a fixed-point multiplication by a factor $m = (m_1 \cdot m_2)/m_3$, borrowing the notation from Section 2.3.2. Recall that this product was handled by writing $m = 2^{-n-31} \cdot m'$, where m' is a 32-bit integer. The model owner canb either perform this decomposition locally for each dot product of the network, and then share the values m' and n with the parties, or the parties can compute these with a secret floating-point division [1].

Separately, the input owner secret-shares his input as floating-point number, after which the parties compute the quantization, again with secret floating-point computation. See Figure 3 for a visualization of the input model.

3.2.1 Secure Computation of a Quantized Dot Product

In this section we show how to compute securely the expression in Eq. (1). Given the setting we described above, here the parties have shares of the zero points z_1, z_2, z_3, the quantized inputs a_i, b_i for $i = 1, \ldots, N$, the integer scale m' and the power 2^n, where $2^{-n-31} \cdot m' \approx m = (m_1 \cdot m_2)/m_3$.

In order to compute the expression in Eq. (1), the parties begin by computing the dot product $\langle s \rangle = \sum_{i=1}^{N} (\langle a_i \rangle - \langle z_1 \rangle) \cdot (\langle b_i \rangle - \langle z_2 \rangle)$, which can be done at the cost of a single secure multiplication as explained in Section 3.1. Then, an additional secure multiplication is used in order to compute $\langle m \cdot s \rangle = \langle m \rangle \cdot \langle s \rangle$. Next, shares of $\lfloor 2^{-n-31} \cdot (m \cdot s) \rceil$ are computed from $\langle n \rangle$ and $\langle m \cdot s \rangle$ using the method described in the next section. Finally,

[7]Notice that these values are only 8-bit long in the clear, but the shares are 64-bit long. The reason is that, although the values are small, the computation must be carried without overflow. Therefore we cannot use a modulus that is smaller than the maximum possible intermediate value.

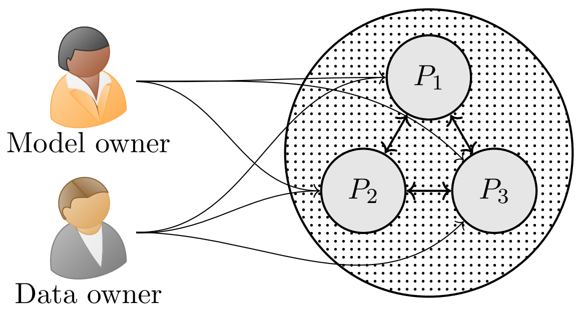


Figure 3: Visualization of the Client-Server model we consider in this work. The model and data owner secret-share their data towards the three servers, who then execute the secure computation and return the result to the clients.

addition with $\langle z_3 \rangle$ is local, and it is followed by the clamping method described in Section 3.2.5.

3.2.2 Truncation by a Secret Shift

Assume the parties have shared values $\langle x \rangle$ and $\langle k \rangle$ for $k \leq K$ for some public K, and they wish to compute shares of $\lfloor 2^{-k} \cdot x \rceil$. Trivially, $2^{-k} \cdot x = 2^{K-k} \cdot 2^{-K} \cdot x$. We use this because truncation by a public number of bits is more efficient to achieve. 2^{K-k} can computed from a bit decomposition $K - k = \sum_i b_i \cdot 2^i$ as $\prod_i (1 + b_i \cdot (2^{2^i} - 1))$. It remains to compute the public shift below.

3.2.3 Truncation by a Public Shift

Assume the parties have shared value $\langle x \rangle$ and public k, and they wish to compute shares of $\lfloor 2^{-k} \cdot x \rceil$. Note that $\lfloor 2^{-k} \cdot x \rceil = \lfloor 2^{-k} \cdot x + 0.5 \rfloor = \lfloor 2^{-k} \cdot (x + 2^{k-1}) \rfloor$ for breaking a tie by rounding up. It therefore suffices to compute $\lfloor 2^{-k} \cdot x \rfloor$, that is plainly shifting to the right. First compute $\langle x' \rangle = \langle x \rangle \bmod 2^k$ as explained below. $\langle x - x' \rangle$ is then guaranteed to have the k least significant bits set to zero. We proceed by masking the top-most $64 - k$ bits, that is, computing $\langle x - x' \rangle + 2^k \sum_{i=0}^{64-k} \langle b_i \rangle \cdot 2^i$, for fresh random bits $\langle b_i \rangle$.[8] Because of the wrap-around modulo 2^{64} and the fact that the k least significant bits of $x - x'$ are 0, the masked number does not reveal any information and can thus be revealed as c. We can then shift c to the right and undo the addition of $\sum_{i=0}^{64-k} \langle b_i \rangle \cdot 2^i$. However, this does not undo the potential wrap-around. We therefore have to compute the comparison of $c \gg k$ and $\sum_{i=0}^{64-k} \langle b_i \rangle \cdot 2^i$ as a binary circuit and subtract 2^{64-k} from the result if necessary.

3.2.4 Modulo a Public Power of Two

Assume the parties have shared value $\langle x \rangle$ and public k, and they wish to compute shares of $\langle x \bmod 2^k \rangle$. This can be computed using a similar building blocks as in the last section. We start by computing $x \ll (64 - k)$, efficitively erasing the bits of x that do not form part of the result. Then, we can mask the remaining bits using k random bits, reveal the result, shift the public number to the right. As above we need to compute a binary comparison in order to account for the wrap-around.

[8]These can be simply computed as XOR of two random bits input by separate parties because we assume that they follow the protocol.

3.2.5 Clamping

For the final operation the parties hold $\langle x \rangle$ and need to compute $\langle y \rangle$ where $y = \mathsf{clamp}_{0,2^8-1}(x)$. This is done by comparing $\langle x \rangle$ to the limits (0 and 255), followed by oblivious selection: If $s \in \{0, 1\}$, it holds trivially that $a_s = s \cdot (a_1 - a_0) + a_0$ for arbitrary a_0, a_1.

3.2.6 Comparison

Assume the parties have shared values $\langle x \rangle$ and $\langle y \rangle$ and would like to compute a bit $\langle c \rangle$ such that $c = x \overset{?}{<} y$. Clearly, $c = x - y \overset{?}{<} 0$. We therefore reduce the problem to extraction of the most significant bit of a secret value. This is a special case of the bit decomposition below.

3.2.7 Bit decomposition

Assume the parties have a shared value $\langle x \rangle$ and public k and would like to compute the k most significant bits $\langle x_{63} \rangle, \ldots, \langle x_{64-k} \rangle$ of x. We first set to other bits to 0 by computing $\langle x - x \bmod 2^{64-k} \rangle$ using the algorithm above. Then, we mask the relevant bits with random bits and reveal $x - x \bmod 2^{64-k} - 2^{64-k} \cdot \sum_i b_i 2^i$. This allows us to compute $(x - \sum_i b_i 2^i) \bmod 2^{64-k}$. Finally, we use a binary adder to add the bit decomposition to the random bits, which will result in the desired bits. Note that for the comparison, $k = 1$, which means that the binary adder is simply one XOR (computed as $a + b - 2ab$).

3.2.8 Average and Max Pooling

Average pooling involves computing $\langle y \rangle$ from $\langle x_1 \rangle, \ldots, \langle x_n \rangle$, where $y = \left\lfloor \frac{1}{n} \cdot \sum_{i=1}^{n} x_i \right\rceil$. This can be achieved using Goldschmidt's algorithm [23], a widely used iterative algorithm for division. For its usage in the context of secure multiparty computation, see for example Catrina and Saxena [10]. It uses basic arithmetic as well as truncation, both of which we have explained above.

On the other hand, max pooling requires implementing the max function securely, which can be done by making use of a secure comparison protocol as described above.

3.2.9 Output Layer

Once shares of the output vector are obtained (raw output, before applying Softmax), several options can be considered. The parties could open the vector itself towards the input owner and/or data owner so that they compute the Softmax function and therefore learn the probabilities for each label. However, this would reveal all the prediction vector, which could be undesirable in some scenarios. Thus, we propose instead to securely compute the argmax of the output array, and return this index, which returns the most likely label since exponentiation is a monotone increasing function.

It is important to remark that Mohassel and Zhang [40] replace the exponentiation in the Softmax function with ReLU operations, i.e. by computing $\mathsf{ReLU}(x)$ instead of e^x. More MPC friendly solutions exist, such as the spherical Softmax [16], which replaces e^x with x^2.

4 Implementation and Benchmarking

4.1 MobileNets Architecture

The *MobileNet* line of models [50, 29] from Google has shown much promise for performing image recognition related tasks (object detection, segmentation etc.) on computationally constrained devices, such as smartphones or embedded devices. Their widespread use is helped by the fact that they are fully supported by TensorFlow lite and that several variants are readily available online already pretrained on the ImageNet dataset. Moreover, these pretrained models exist as both floating point and quantized versions.[9] In Section A.1 in the appendix we present the different blocks used in the MobileNets networks, which also serves as a short description of the various optimizations that Tensorflow performs as part of the quantization process (some of which was already hinted at in Section 2.3.3). In the following, MobileNetV1 will refer to the model by Howard et al. [29], and MobileNetV2 will refer to the model by Sandler et al. [50].

4.1.1 Network Structure

The structure of a MobileNetsV1 network is fairly straightforward: the input layer is a regular convolution (and in fact the only such one). This layer is immediately followed by 13 depthwise separable convolutions, i.e. 13 alternating depthwise and pointwise convolutions. The last two layers are an average pool followed by a fully connected layer, for a total of 28 layers (not counting a softmax at the end for turning the result into a probability distribution).

The architecture contains two hyper parameters that allow the user to scale the network in different ways, namely a *width multiplier* α and a *resolution multiplier* ρ. The width multiplier scales the input and output channels. I.e., the number of multiply-add instructions in each convolution becomes $\alpha I_d \cdot I_h \cdot I_w \cdot W_h \cdot W_w \cdot \alpha O_d$: the α essentially serves to thin the network. Note that α also affects the number of parameters and thus the model size. The resolution parameter on the other hand simply scales the input; at $\rho = 1.0$ the input is set to be 224×224 and pretrained networks exist for $\rho \approx 0.85$ (192×192), $\rho \approx 0.71$ (160×160) and $\rho \approx 0.57$ (128×128). Note that scaling ρ affects the number of multiply-add operations, but does not affect the size of the network.

4.2 Benchmarks

We have run a number of different benchmarks, providing evidence that it is indeed possible to run realistically sized models, in particular the MobileNetsV1 models for different choice of the two hyperparameters. Besides providing benchmarks for a computation comparable to evaluating a full network, we also investigate, through micro-benchmarks, the effect the number of dot products has on the running time and communication, as well as the effect the length of the dot products have. Perhaps not surprisingly, we find that communication is independent of the length of each dot product (due to the nature of the sum-of-product optimization), and scales essentially linearly with the number. Also not surprisingly, we find the running time scales linearly with the number of dot products, with 200 000 taking less than a second (for comparison, the smallest MobileNetV1 model only has around 400 000 in total).

In addition to running all our experiments over the ring $\mathbb{Z}_{2^{64}}$, we also ran experiments over a prime field, showing first that $\mathbb{Z}_{2^{64}}$ is indeed the optimal choice for passive security,

[9] `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/g3doc/models.md#image-classification-quantized-models`

as it is orders of magnitudes faster. We additional show results of running the same operations in a protocol (also in a prime field) with *active* security, and observe that this too is within realistic bounds. In particular, this is the first concrete demonstration that evaluating a CNN with active security is feasible.

We ran all our benchmarks on co-located c9.5xlarge AWS machines, each of which has 36 cores, 72gb of memory and a 10gpbs link between them. Throughout this section, communication is measured per party and all timings include preprocessing.

Micro-benchmarks. Our micro-benchmarks are focused first and foremost on measuring the cost, in terms of time and communication, of the core operation of any CNN: the sum-of-product operation (in the MobileNets models, essentially all computations are convolutions). Table 1 shows the result of running a variable number of dot products each of a fixed length, and Table 2 shows the result of running a fixed number of dot products with variable length. We choose numbers that reflect realistic sizes for the convolutions, for example, the largest convolution in the smallest MobileNetsV1 network contains some 60K dot products.

# sum-of-products	mod 2^{64}		mod p		mod p (active security)	
	Runtime (s)	Comm. (gb)	Runtime (s)	Comm. (gb)	Runtime (s)	Comm. (gb)
50 000	0.25	0.15	1.6	0.54	8.8	4.3
100 000	0.41	0.31	2.5	1.07	15.6	8.5
150 000	0.57	0.46	3.6	1.59	22.5	12.8
200 000	0.73	0.62	4.5	2.12	29.2	17.0

Table 1: Results from running a variable number of sum-of-products, each with $\ell = 512$ terms.

# of terms	mod 2^{64}		mod p		mod p (active security)	
	Runtime (s)	Comm. (gb)	Runtime (s)	Comm. (gb)	Runtime (s)	Comm. (gb)
256	0.27	0.31	1.9	1.1	9.4	5.7
512	0.30	0.31	2.0	1.1	13.9	8.5
768	0.33	0.31	2.3	1.1	18.5	11.4
1024	0.36	0.31	2.4	1.1	22.9	14.3

Table 2: Running $n = 100.000$ sum-of-products with variable length.

Prime order field Due to capabilities of MP-SPDZ, we could also run our benchmarks in replicated secret sharing modulo a 128-bit prime, both with semi-honest and malicious security [20]. For example, variant 0.50_128 (width multiplier 0.5, input size 128×128) of V1 takes 15 seconds with semi-honest security and 48 seconds with malicious security. This shows that using $\mathbb{Z}_{2^{64}}$ as the computation domain is preferable to prime order fields. We consistently found at least a five-fold improvement in time and a three-fold improvement in communication.

Full model evaluation We ran benchmarks whose sizes correspond to a full network evaluation; for these, we construct a program which in a number of iterations perform the number of sum-of-products that would be needed for a particular layer in a particular network. Results can be seen in Table 3. For each model we ran a benchmark using the protocol over $\mathbb{Z}_{2^{64}}$, and over a prime field with both passive and active security. The results we obtain heavily imply that, for passive security, $\mathbb{Z}_{2^{64}}$ is preferable.

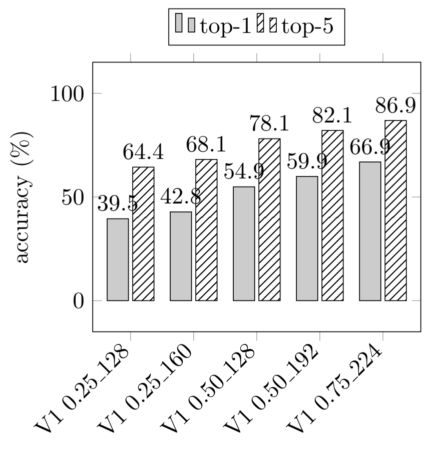
We also ran a test on the single pretrained MobileNetV2, which has width multiplier 1.0 and input size 224×224. This network could be evaluated in around 14 seconds with 20gb of communication. We remark we do not have all the building blocks for this network (missing is the residual connection, cf. Appendix A.2; note that the residual connection is not just a simple pointwise addition because we have to take the quantization parameters into account, and so both scaling and clamping needs to be performed) and so these numbers are only approximate.

Ver.	Variant	mod 2^{64}		mod p		mod p (active security)		Accuracy	
		Runtime (s)	Comm. (gb)	Runtime (s)	Comm. (gb)	Runtime (s)	Comm. (gb)	Top-1 (%)	Top-5 (%)
V1	0.25_128	1.1	1.3	7.1	4.4	22.3	12.9	39.5	64.4
	0.25_160	1.6	2.0	10.9	6.8	34.3	20.1	42.8	68.1
	0.25_192	2.5	2.9	15.7	9.8	49.0	28.9	45.7	70.8
	0.25_224	3.2	3.9	21.2	13.3	66.3	39.3	48.2	72.8
	0.50_128	2.3	2.5	14.0	8.7	47.5	28.2	54.9	78.1
	0.50_160	3.3	4.0	21.7	13.6	73.8	44.0	57.2	80.5
	0.50_192	4.7	5.7	30.9	19.5	105.5	63.3	59.9	82.1
	0.50_224	5.8	7.7	42.2	26.6	143.5	86.1	61.2	83.2
	0.75_128	3.0	3.8	21.0	13.0	76.7	46.0	55.9	79.1
	0.75_160	4.5	5.9	32.3	20.4	119.0	71.7	62.4	83.7
	0.75_192	6.5	8.5	46.2	29.3	171.0	103.2	66.1	86.2
	0.75_224	8.6	11.6	63.6	39.9	232.9	140.5	66.9	86.9
	1.00_128	4.2	5.1	28.3	17.4	109.8	66.2	63.3	84.1
	1.00_160	6.2	7.9	43.3	27.1	170.5	103.3	66.9	86.7
	1.00_192	8.5	11.4	61.9	39.1	244.3	148.7	69.1	88.1
	1.00_224	11.5	15.5	83.9	53.2	332.5	202.4	70.0	89.0

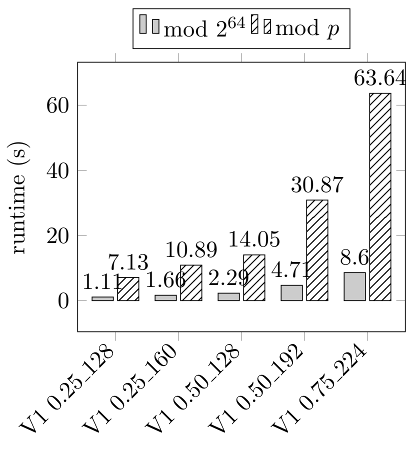
Table 3: Benchmark of the MobileNets family of networks. The first number in variant is the width multiplier and the second is the resolution multiplier. Accuracy numbers taken from `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/lite/g3doc/models.md#image-classification-quantized-models`

Comparison with SecureNN In order to put our protocol into perspective of prior literature in the area of secure evaluation of ML protocols, we extracted the relevant parameters from the four different models used in SecureNN [57] and ran some benchmarks. We remark that only 8 cores of the 36 available was used for these experiments, rather than the full 36 because of the limited potential for further parallelization.

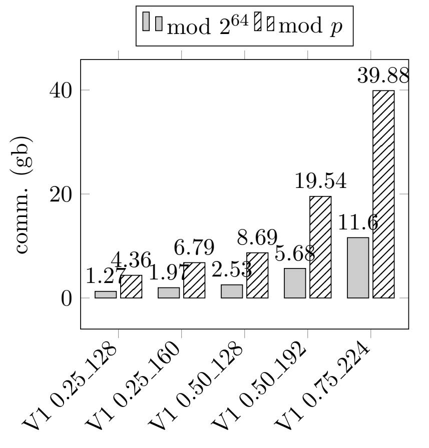
The benchmarks can be seen in Table 4. Interestingly, while our protocol is faster for all models, only in models A and D are we also more efficient in terms of communication. One possible explanation for this, is that networks A and D contain only convolutions (or fully connected layers, both of which only involve dot products); while networks B and C both contain a relatively large amount of maxpool layers. (Both B and C have five layers of which two are maxpool layers). We see however that the difference disappears when the network contains more sum-of-products: while B and C are essentially the same network, the convolutions in network C are larger, and so the difference between the amount of communication in our protocol and the one from SecureNN are smaller.



(a) Accuracy



(b) Runtimes



(c) Communication complexity

Figure 4: Runtime and communication for some of the networks in the MobileNet family.

	Protocol	Runtime (s)	Communication (mb)
A	SecureNN (3PC)	0.05	4.03
	Quantized	0.02	2.41
B	SecureNN (3PC)	0.22	17.28
	Quantized	0.08	33.12
C	SecureNN (3PC)	0.34	37.03
	Quantized	0.12	46.74
D	SecureNN (3PC)	0.10	7.93
	Quantized	0.02	3.73

Table 4: Comparison of running times for our protocol with the 3-party protocol in SecureNN [57].

4.3 Extending to Other TensorFlow Models

TensorFlow already supports quantizing pretrained floating-point models [10] and quantization-aware training, [11] and so one could ask what steps need to be taken in order to go from a model designed and trained in TensorFlow, to secure inference. Since the training framework already exists, then (at least intuitively), the only step missing is connecting a description of a pretrained TensorFlow model to an MPC framework; and if this framework uses the techniques we have shown to be applicable to realistically sized models, then this should provide a way to run these models.

Some engineering aspects needs to be addressed however. Firstly, parsing the models output by TensorFlow is needed. This, however, is fairly straightforward as the generated models are stored in a very well documented binary format, namely as *flatbuffers* in the case for `.tflite` models (which are the ones output by the quantized training process). More arduous is the handling of data, as we here need to ensure that everything can be efficiently and correctly treated as inputs to a sum-of-products routine. This step is complicated by the fact that windows of size larger than 1×1 (which means all depthwise convolutions) require non-trivial indexing. One approach to handling the data between layers, is to ensure that all outputs are formatted as Toeplitz matrices as these allow for convolutions to be computed as matrix products (which is essentially what we need).

5 Conclusions

Our work constitutes, to the best of our knowledge, the first protocol for secure evaluation of quantized neural networks using MPC techniques. We show that secure evaluation without accuracy loss of large, realistic CNNs, like those in the MobileNets family, is within reach. Compared to existing models which have been used for secure inference, we found that our approach faster. However, with respect to communication we find that the networks need to contain a high amount of dot products before the effect of the sum-of-product optimization becomes apparent. This is not an issue for the MobileNets models (as these are essentially *only* dot products) but for smaller models.

We also narrow the gap between evaluating a specific hardcoded model and an arbitrary (Tensorflow) model by using well-known tools both from the ML and MPC domains, such as Tensorflow Lite and the MP-SPDZ compiler, and due to this we believe that a full-fledged architecture for easy deployment of these models in real-life scenarios is already within engineering range.

5.1 Future Work

MPC/FHE-aware ML research. We believe that it is important for researchers in the area of secure computation of ML models to adhere to standard techniques in Machine Learning in order to validate the accuracy of their methods. Quantization is a research area in the ML domain that is motivated by the existence of resource-constrained devices and the necessity of deploying machine learning models in these scenarios. We think that secure computation of such models can also serve as a motivation to push forward research in the machine learning domain, i.e. the study of new ML tools that are MPC/FHE-friendly, instead of the reverse (MPC/FHE protocols that are ML friendly).

[10] `https://www.tensorflow.org/lite/performance/post_training_quantization`

[11] `https://github.com/tensorflow/tensorflow/blob/master/tensorflow/contrib/quantize/README.md`

Other quantization schemes. We expressed in Section 2.3 the benefits of using the quantization scheme by Jacob et al. However, secure inference using quantization is a rich area of research in the machine learning domain, and there are many other quantization schemes that could be also useful for the setting of secure evaluation. Unfortunately, one of the limiting factors in applying other techniques is the fact that in the MPC setting, no party should know anything about the underlying data and model, in contrast to the FHE setting in which the data owner can encrypt its data so that the model owner executes the homomorphic evaluation with its model locally in the clear. This imposes some challenges when using quantization schemes that allow the model to have only ± 1 as weights, so that inference consists mostly of additions and subtractions only. This has been explored very recently in the FHE setting [8, 51], but it is future work to apply these to the MPC setting.

Training. Our protocol is designed in the inference setting, and we leave it as future work to apply quantization techniques in order to realize secure training of quantized CNNs with little-to-no accuracy loss with respect to their floating-point counterparts. It is important to notice that the spectrum of research in quantized training of CNNs is much more reduced than in quantized inference, with only a few recent works like [30], [59] and [61]. Moreover, it is not entirely clear how compatible these methods are with MPC or FHE techniques.

Active security. As we have shown, working modulo a prime is less efficient than modulo a power of 2, but on the other hand it allows us to obtain a protocol with active security. Protocols with active security modulo a power of 2 are not as straightforward, and only recently one has been published for the dishonest majority setting [14]. We conjecture that an active secure version of our protocol modulo 2^{64} would outperform its modulo p counterpart, and we leave it as future work to verify the validity of this claim.

Acknowledgements

This work has been supported by the European Research Council (ERC) under the European Unions's Horizon 2020 research and innovation programme under grant agreement No 669255 (MPCPRO), the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS), by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Directorate in the Prime Minister's Office, and by the Alter Family Foundation, and the Danish Independent Research Council under Grant-ID DFF-6108-00169 (FoCC) and the European Union's Horizon 2020 research and innovation programme under grant agreement No 731583 (SODA),

We thank Adrià Gascón for fruitful discussions, and also NEC and BIU members, especially Prof. Benny Pinkas and Prof. Yehuda Lindell for very useful comments on this work.

A Appendix

A.1 MobileNets Blocks

We present below some of the different blocks present in the MobileNets family of networks.

Depthwise Separable Convolutions. The majority of the computation that a CNN performs, and the space that it uses, is tied to its convolutional operations. A regular convolution performs at its core two steps: first it filters the input using a set of trained weights, by moving each filter over the entire input; and second, a convolution combines the output of each filter application to produce a single output value. The whole operation can be viewed as an entry-wise product between a filter and each input in a specific window, followed by a summation of all the products. The price of a convolution is therefore $I_d \cdot I_h \cdot I_w \cdot W_h \cdot W_w \cdot O_d$. What the MobileNets models does instead, is to replace these convolutions with a *depthwise separable convolution.* At a high level, the idea behind a depthwise separable convolution is to split the two tasks outlined above into to separate operations.

More precisely, instead of performing a normal convolution, first a depthwise convolution is performed, which is like a regular convolution except it does not change the output depth; afterwards, a *pointwise* convolution is performed. A pointwise is a regular convolution with a 1×1 filter, which preserves the input dimensions but allows for scaling of the depth. The cost of the depthwise convolution is $I_h \cdot I_w \cdot I_d \cdot W_w \cdot W_h$ while the cost of the pointwise convolution is $O_d \cdot I_d \cdot I_h \cdot I_w$. Replacing a normal regular convolution with a Depthwise Separable convolution provides a saving of $1/O_d + 1/(W_h \cdot W_w)$ in terms of computation.

In MobileNetsV1, both the depthwise and pointwise convolution are followed by a batch normalization layer, and a ReLU6 activation.

Batch Normalization. Batch normalization [31] is a technique used to speed up training by normalizing the inputs to each activation: instead of computing $g(x)$ for some input x and activation g, we instead compute $g(y)$ where

$$y = \gamma \left(\frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \right) + \beta,$$

where γ, β are parameters learned during training, and μ_B, σ_B^2 is the mean and variance, respectively, of a batch B of which x is a member. During inference the same scaling applies except the mean and variance are those of the entire training set (which are learned during training). However, because the μ and σ used during inference are constant it is possible to completely "fold" the batch normalization scaling and shifting into the parameters of the model. More precisely, for an input $y = xW + b$ to the activation function g, where we need to compute $y' = \gamma((y - \mu)/\sigma) + \beta$ (ignoring ϵ), we can instead use different weights W' and bias b' defined as

$$w' = \frac{\gamma W}{\sigma}, \qquad\qquad b' = \gamma \left(\frac{b - \mu}{\sigma} \right) + \beta.$$

Note that

$$xW' + b' = x\frac{\gamma W}{\beta} + \gamma \left(\frac{b - \mu}{\sigma} \right) + \beta = \gamma \left(\frac{y - \mu}{\sigma} \right) + \beta,$$

as required. That is, the Batch Normalization operation disappears during inference.

ReLU6. As mentioned already in Section 2.3.3, the ReLU6 operation can be computed as part of the clamping when quantization is used and so this operation is also not explicit in the quantized version of a MobileNets model. (It is, however, needed in the floating point variant.)

A.2 MobileNets V2

The MobileNetsV2 architecture shares many of the same characteristics described above. Here as well, the idea of replacing regular convolutions with depthwise separable convolutions is used. However, the authors alter the design of this block by placing pointwise convolution before the depthwise convolution. The idea is to have the first pointwise convolution expand the input (by some factor t), then apply the depthwise convolution, and then finally shrink the input by t using another pointwise convolution. The first pointwise convolution and the depthwise convolution is followed by a ReLU6 activation, while the last uses the identity function (i.e. no activation). This operation is referred to as a *bottleneck*.

The other addition to the architecture is the use of residual connections, where the input of a bottleneck block is added to the output. That is, if $f(x)$ computes a bottleneck operation on the input x, then a residual bottleneck operation would compute $f(x) + x$. See Figure 5 for a visualization of a part of the network.

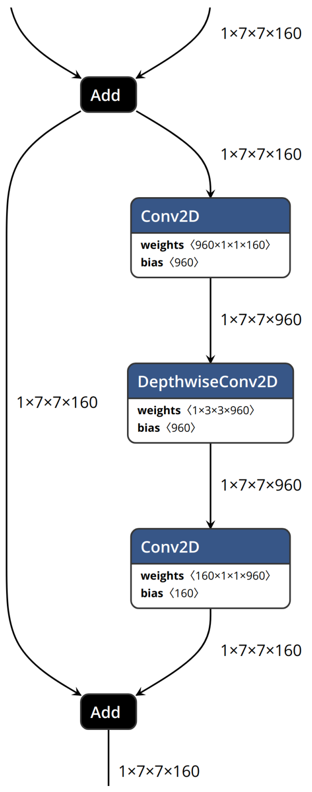


Figure 5: Excerpt from the MobileNets V2 Network, illustrating the use of residual connections. This pattern appears several times in the network (with different sizes). The network has been visualized using Netron [48].

Otherwise the architecture remains the same: the input is first passed through a regular convolution, then a number of bottleneck blocks, followed in the end by an average pool and classification.

References

[1] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure computation on floating point numbers. In *ISOC Network and Distributed System Security Symposium –*

NDSS 2013, San Diego, CA, USA, Feb. 24–27, 2013. The Internet Society.

[2] T. Araki, A. Barak, J. Furukawa, M. Keller, Y. Lindell, K. Ohara, and H. Tsuchida. Generalizing the SPDZ compiler for other protocols. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 18: 25th Conference on Computer and Communications Security*, pages 880–895, Toronto, ON, Canada, Oct. 15–19, 2018. ACM Press.

[3] T. Araki, A. Barak, J. Furukawa, T. Lichter, Y. Lindell, A. Nof, K. Ohara, A. Watzman, and O. Weinstein. Optimized honest-majority MPC for malicious adversaries - breaking the 1 billion-gate per second barrier. In *2017 IEEE Symposium on Security and Privacy*, pages 843–862, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[4] W. Balzer, M. Takahashi, J. Ohta, and K. Kyuma. Weight quantization in boltzmann machines. *Neural Networks*, 4(3):405–409, 1991.

[5] M. Barni, C. Orlandi, and A. Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th workshop on Multimedia and security*, pages 146–151. ACM, 2006.

[6] D. Beaver. Efficient multiparty protocols using circuit randomization. In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432, Santa Barbara, CA, USA, Aug. 11–15, 1992. Springer, Heidelberg, Germany.

[7] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016.

[8] F. Bourse, M. Minelli, M. Minihold, and P. Paillier. Fast homomorphic evaluation of deep discretized neural networks. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 483–512, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany.

[9] O. Catrina and S. de Hoogh. Improved primitives for secure multiparty integer computation. In J. A. Garay and R. D. Prisco, editors, *SCN 10: 7th International Conference on Security in Communication Networks*, volume 6280 of *Lecture Notes in Computer Science*, pages 182–199, Amalfi, Italy, Sept. 13–15, 2010. Springer, Heidelberg, Germany.

[10] O. Catrina and A. Saxena. Secure computation with fixed-point numbers. In R. Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 35–50, Tenerife, Canary Islands, Spain, Jan. 25–28, 2010. Springer, Heidelberg, Germany.

[11] H. Chabanne, A. de Wargny, J. Milgram, C. Morel, and E. Prouff. Privacy-preserving classification on deep neural network. Cryptology ePrint Archive, Report 2017/035, 2017. `http://eprint.iacr.org/2017/035`.

[12] M. Courbariaux and Y. Bengio. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016.

[13] 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*, pages 3123–3131, 2015.

[14] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for dishonest majority. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 769–798, Santa Barbara, CA, USA, Aug. 19–23, 2018. Springer, Heidelberg, Germany.

[15] R. Cramer, I. Damgård, and Y. Ishai. Share conversion, pseudorandom secret-sharing and applications to secure computation. In J. Kilian, editor, *TCC 2005: 2nd Theory of Cryptography Conference*, volume 3378 of *Lecture Notes in Computer Science*, pages 342–362, Cambridge, MA, USA, Feb. 10–12, 2005. Springer, Heidelberg, Germany.

[16] A. de Brébisson and P. Vincent. An exploration of softmax alternatives belonging to the spherical loss family. *arXiv preprint arXiv:1511.05042*, 2015.

[17] D. Demmler, T. Schneider, and M. Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*, San Diego, CA, USA, Feb. 8–11, 2015. The Internet Society.

[18] A. Esteva, B. Kuprel, R. A. Novoa, J. Ko, S. M. Swetter, H. M. Blau, and S. Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115, 2017.

[19] E. Fiesler, A. Choudry, and H. J. Caulfield. Weight discretization paradigm for optical neural networks. In *Optical interconnections and networks*, volume 1281, pages 164–174. International Society for Optics and Photonics, 1990.

[20] J. Furukawa, Y. Lindell, A. Nof, and O. Weinstein. High-throughput secure three-party computation for malicious adversaries and an honest majority. In J. Coron and J. B. Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part II*, volume 10211 of *Lecture Notes in Computer Science*, pages 225–255, Paris, France, Apr. 30 – May 4, 2017. Springer, Heidelberg, Germany.

[21] A. Gascon, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Secure linear regression on vertically partitioned datasets. Cryptology ePrint Archive, Report 2016/892, 2016. `http://eprint.iacr.org/2016/892`.

[22] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 201–210, 2016.

[23] R. E. Goldschmidt. Applications of division by convergence. Master's thesis, MIT, 1964.

[24] Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[25] Y. Guo. A survey on methods and theories of quantized neural networks. *arXiv preprint arXiv:1808.04752*, 2018.

[26] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[27] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[28] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[29] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[30] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[31] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pages 448–456, 2015.

[32] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. G. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *CoRR*, abs/1712.05877, 2017.

[33] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018.*, pages 1651–1669, 2018.

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

[35] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[36] S. Lawrence, C. Lee Giles, A. Chung Tsoi, and A. Back. Face recognition: A convolutional neural network approach. *Neural Networks, IEEE Transactions on*, 8:98 – 113, 02 1997.

[37] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via MiniONN transformations. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 17: 24th Conference on Computer and Communications Security*, pages 619–631, Dallas, TX, USA, Oct. 31 – Nov. 2, 2017. ACM Press.

[38] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini. Fast neural networks without multipliers. *IEEE transactions on Neural Networks*, 4(1):53–62, 1993.

[39] T. Mikolov, I. Sutskever, A. Deoras, H.-S. Le, S. Kombrink, and J. Cernocky. Subword language modeling with neural networks. *preprint (http://www. fit. vutbr. cz/imikolov/rnnlm/char. pdf)*, 8, 2012.

[40] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*, pages 19–38, San Jose, CA, USA, May 22–26, 2017. IEEE Computer Society Press.

[41] N1 Analytics. MP-SPDZ. `https://github.com/n1analytics/MP-SPDZ`, 2018.

[42] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. GraphSC: Parallel secure computation made easy. In *2015 IEEE Symposium on Security and Privacy*, pages 377–394, San Jose, CA, USA, May 17–21, 2015. IEEE Computer Society Press.

[43] C. Orlandi, A. Piva, and M. Barni. Oblivious neural network computing via homomorphic encryption. *EURASIP Journal on Information Security*, 2007(1):037343, 2007.

[44] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[45] E. Park, J. Ahn, and S. Yoo. Weighted-entropy-based quantization for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7197–7205. IEEE, 2017.

[46] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, pages 525–542, 2016.

[47] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar. Chameleon: A hybrid secure computation framework for machine learning applications. In J. Kim, G.-J. Ahn, S. Kim, Y. Kim, J. López, and T. Kim, editors, *ASIACCS 18: 13th ACM Symposium on Information, Computer and Communications Security*, pages 707–721, Incheon, Republic of Korea, Apr. 2–6, 2018. ACM Press.

[48] L. Roeder. Netron, visualizer for deep learning and machine learning models. `https://github.com/lutzroeder/Netron`, 2019. Accessed: Feb 5, 2019.

[49] B. D. Rouhani, M. S. Riazi, and F. Koushanfar. Deepsecure: scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, pages 2:1–2:6, 2018.

[50] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 4510–4520, 2018.

[51] A. Sanyal, M. J. Kusner, A. Gascón, and V. Kanade. Tapas: Tricks to accelerate (encrypted) prediction as a service. *arXiv preprint arXiv:1806.03461*, 2018.

[52] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[53] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[54] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[55] C. Z. Tang and H. K. Kwan. Multilayer feedforward neural networks with single powers-of-two weights. *IEEE Transactions on Signal Processing*, 41(8):2724–2727, 1993.

[56] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. Stealing machine learning models via prediction apis. In *USENIX Security Symposium*, pages 601–618, 2016.

[57] S. Wagh, D. Gupta, and N. Chandran. SecureNN: Efficient and private neural network training. Cryptology ePrint Archive, Report 2018/442, 2018. `https://eprint.iacr.org/2018/442`.

[58] B. Wang and N. Z. Gong. Stealing hyperparameters in machine learning. In *2018 IEEE Symposium on Security and Privacy*, pages 36–52, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[59] N. Wang, J. Choi, D. Brand, C.-Y. Chen, and K. Gopalakrishnan. Training deep neural networks with 8-bit floating point numbers. In *Advances in neural information processing systems*, pages 7686–7695, 2018.

[60] Wikipedia. Lee sedol. `https://en.wikipedia.org/wiki/Lee_Sedol`. Accessed: 19-12-2018.

[61] S. Wu, G. Li, F. Chen, and L. Shi. Training and inference with integers in deep neural networks. *arXiv preprint arXiv:1802.04680*, 2018.

[62] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016.