ABSTRACT

Privacy concerns in client-server machine learning have given rise to private inference (PI), where neural inference occurs directly on encrypted inputs. PI protects clients’ personal data and the server’s intellectual property. A common practice in PI is to use garbled circuits to compute nonlinear functions privately, namely ReLUs. However, garbled circuits suffer from high storage, bandwidth, and latency costs. To mitigate these issues, PI-friendly polynomial activation functions have been employed to replace ReLU. In this work, we ask: Is it feasible to substitute all ReLUs with low-degree polynomial activation functions for building deep, privacy-friendly neural networks? We explore this question by analyzing the challenges of substituting ReLUs with polynomials, starting with simple drop-and-replace solutions to novel, more involved replace-and-retrain strategies. We examine the limitations of each method and provide commentary on the use of polynomial activation functions for PI. We find all evaluated solutions suffer from the escaping activation problem: forward activation values inevitably begin to expand at an exponential rate away from stable regions of the polynomials, which leads to exploding values (NaNs) or poor approximations.

Table 1: Effectiveness of polynomial activations proposed for ciphertext training/inference on MNIST, CIFAR-10 (C10), CIFAR-100 (C100), and Tiny-ImageNet (Tiny) datasets.

| Method          | Degree | Partial/Full | Datasets |
|-----------------|--------|--------------|----------|
|                 |        |              | MNIST   | C10 | C100 | Tiny |
| CryptoNet [16]  | 2      | Full         | Y       | N   | N    | N    |
| Lookup Table [52] | 2  | Full         | Y       | N   | N    | N    |
| Polyfit [5]     | 2, 4, 6| Full         | Y       | N   | N    | N    |
| SecureML [37]   | 2      | Full         | Y       | N   | N    | N    |
| FCryptoNet [9]  | 2      | Full         | Y       | N   | Y    | N    |
| CryptoDL [20]   | 2      | Full         | Y       | Y   | N    | N    |
| HCNN [2]        | 2      | Full         | Y       | Y   | Y    | N    |
| DELPHI [34]     | 2      | Partial      | N       | Y   | Y    | Y    |
| SAFENet [32]    | 2      | Partial      | N       | Y   | Y    | Y    |
| PreciseApprox [30] | 29  | Full         | Y       | N   | Y    | Y    |
| QuaIL (Ours)    | 2      | Full         | Y       | Y   | Y    | Y    |

1 INTRODUCTION

The growing adoption of Machine Learning as a Service (MLaaS) [22] has given rise to privacy concerns of clients’ personal data and the intellectual property (i.e., trained models) of service providers. To address these concerns, techniques such as different privacy-preserving machine learning (PPML) at-studies within privacy-preserving machine learning (PPML) attempts to perform inference directly on encrypted data using either HE [16, 37, 47] or MPC-based techniques such as Secret Sharing (SS) [6, 7, 36, 40, 41, 44–46]. Common PI protocols employ HE/SS for processing linear operations (e.g., convolutions and fully connected layers) and garbled circuits for nonlinear operations (e.g., ReLU and maxpool) [23, 24, 31, 32, 34, 43].

Garbled circuits are a major source of inefficiency when performing PI for the following reasons: (1) PI, unlike plaintext inference, ReLU garbled circuits dominate the runtime and can be orders of magnitude more costly than linear layers computed with SS [15, 34]; (2) a single ReLU operation using garbled circuits requires 17.5 KB of data storage and communication, and a single inference on state-of-the-art DNNs (such as ResNet50 [18]) requires millions of ReLU computations that leads to hundreds of GiB of data storage and communication [43]. These inefficiencies exist for variants of ReLU such as leaky ReLU [33], parametric ReLU [17], RReLU [54], CReLU [49], and the recently proposed DY-ReLU [17]. Furthermore, storage and latency costs of GCs are exacerbated when used to compute more expressive and complex activation functions such as ELU [10], SELU [25], Swish [42], GELU [19] and Mish [35].

The aforementioned challenges and inefficiencies of nonlinear computations using garbled circuits have driven researchers to design alternative activation functions that are cheaper to compute under PI. In particular, polynomial functions, which require only simple addition and multiplication, eliminate the need for garbled circuits and have become the de-facto solution for replacing ReLUs in neural networks. In fact, replacing all ReLUs with $x^2$ (denoted Quad here) can reduce online latency and communication dramatically by up to 2843× and 256×, respectively [34].

Table 1 summarizes prior work using polynomial activation functions for PI. The partial/full distinction indicates whether the solution replaces some or all ReLU activations with polynomials. We find that prior work can be classified into three categories: full replacement using small datasets/models (e.g., MNIST [29], CIFAR-10 [27]) [2, 5, 9, 16, 20, 37, 52], partial replacement on mid-sized models (e.g., CIFAR-100) [32, 34], and full-replacement on large models using very-high degree approximations [30]. Each of the solutions significantly advanced our understanding of the problem and the capabilities of PI. However, none have demonstrated full replacement on large datasets/models using low-degree polynomials, which we believe is the ideal solution.

In this paper, we set out to replace all ReLUs with low-degree polynomials. Specifically, we test two drop-and-replace solutions (Taylor Approximation and Polynomial Regression Approximation) and develop two novel replace-and-retrain strategies (QuaIL and
We test the Sisyphus framework on the MNIST, CIFAR-10, and CIFAR-100, and TinyImageNet [55] datasets and test each substitution strategy over a wide variety of networks: AlexNet [28], VGG11/VGG16 [50], ResNet18, MobileNetV1 [21], and ResNet32 [18]. We develop and test our framework using PyTorch [39] (1.8.1 + CUDA11.1), and for performing Bayesian Optimization during Polynomial Regression Approximation, we utilize GPyTorch [13], and BoTorch [3].

Our contribution can be summarized as follows:

1. We propose Quadratic Imitation Learning (QuaIL), a training setup inspired by dynamic programming to gradually build neural networks with only polynomial activations and introduce ApproxMinMaxNorm, a normalization strategy that bounds pre-activation values during training and approximately bounds pre-activation values during inference.

2. We implement and release Sisyphus, a set of methods for wholesale ReLU replacement that range from simple drop-and-replace solutions to replace-and-retrain strategies.

3. We develop and rigorously evaluate four substitution strategies using the Sisyphus framework and perform an in-depth analysis of their efficacy for deep networks. Crucially, we show that the instabilities of performing both inference and training with polynomial activation functions become more prominent in deeper neural networks and may not be observed in shallower networks.

As we increase the complexity of the replacement strategy we steadily progress towards training deeper, more accurate, PI-friendly networks using only low-degree polynomial activations. Despite our best efforts, we fall short of matching baseline ReLU network performance due to the escaping activation problem: in all solutions, forward-pass activation values inevitably escape the well-behaving range of the polynomial activation function, leading to either exploding values (NaNs) or poor-behaving approximations.

Looking beyond QuaIL+ApproxMinMax (QuaIL+AMM), it may be tempting to evaluate additional solutions. One way to overcome the escaping activation problem in QuaIL would be to bound the range. However, this requires a max function, which if we had, we could simply use to compute ReLU in our networks. Recent work proposed the Padé Activation Unit (PAU), a rational function of two low-degree polynomials that performs well on complex datasets [38]. Unfortunately, the division operation required by PAUs is not natively supported by cryptographic primitives of HE/SS and is known to be a challenge to implement. Another recent work has proposed approximating ReLU and max-pooling using very-high (e.g., 29) degree polynomials and reports competitive accuracy for ImageNet [30]. However, high-degree polynomials can be difficult to evaluate using cryptographic solutions as they would introduce significant additional computation in both SS and HE as well as noise growth in HE. Thus, we name this paper and our framework Sisyphus, as each time a promising solution was found by minimizing the mean squared error between the approximation and the target function

\[ E(\tilde{\omega}) = \int_{-a}^{a} \left( \sum_{d=0}^{D} w_d x^d - \text{ReLU}(x) \right)^2 dx. \]  

\[ \text{(1)} \]

Setting: First, a baseline ReLU model is trained. We then replace all ReLUs in the trained networks with the Taylor approximation and measure the test accuracy for the network’s respective dataset.

Results: As evident in Table 2, the test accuracy deteriorates significantly for all networks except for the two layer MLP, which sees a dip in test accuracy from 97.98% (using ReLU) to 86.28% (using the Taylor approximation) on MNIST. Given that the Taylor approximation for ReLU is a simple linear function \( f(x) = \frac{1}{2} x \), we expect deeper networks to perform poorly when using the approximation as an activation function.

Takeaway: Using the Taylor approximation of ReLU collapses each network to a linear model, which restricts the network from representing the non-linear mappings required to achieve a high predictive performance on deeper networks and complex datasets.

3.1.2 Polynomial Regression Approximation.

Key Idea: A natural extension of the Taylor approximation to approximate ReLU is to use a polynomial over a range rather than a single point. The polynomial fit to a function \( f \) has the form \( f(x) = \tilde{\omega} \cdot \tilde{x} \), where \( \tilde{x} = (1, x^1, x^2, \ldots, x^D) \) and \( D \) is the order of the polynomial. Polynomial regression can be employed to fit a polynomial function to any non-linear function by minimizing the mean squared error between the approximation and the target function \( f \) over a range \([-a, a]\) and order \( D \). For example if the target function is ReLU, optimal coefficients \( w_0, w_1, \ldots, w_D \) can be found by minimizing

\[ E(\tilde{\omega}) = \int_{-a}^{a} \left( \sum_{d=0}^{D} w_d x^d - \text{ReLU}(x) \right)^2 dx. \]  

\[ \text{(1)} \]

Setup: To find \( \tilde{\omega} \) that minimizes Equation 1, we first discretize the integral using a granularity of \( dx = 1e^{-3} \). The polynomial fit heavily depends upon the order of the polynomial \( (D) \) and the range \( (a) \) over which Equation 1 is minimized. To this end, we employ Bayesian Optimization (BayesOpt) to efficiently select effective values for \( D \) and \( a \) [51]. To accommodate a variety of polynomials, we choose the range \( a \) to vary between \([0.5, 50]\) and the order of the polynomial to vary over integer values between \([2, 9]\).

Given a setting of \((D, a)\), a ReLU approximation is found using polynomial regression. All ReLUs in the original trained network are then replaced with the approximation and we measure the training accuracy. BayesOpt uses this accuracy to iteratively update
Figure 1: An overview of the QuaIL setup for a simple three-layer network. 1) Train a baseline network with the ReLU activation function. 2) Clone the first layer of the ReLU network, copy over the trained weights, and replace ReLU with Quad. Minimize the M.S.E. loss between the first-layer intermediate representations of the two networks and backpropagate through the Quad network. 3) Repeat this process for each subsequent layer (while freezing the previous layers) until the full baseline network is cloned. 4-5) Fine-tune the Quad network by gradually unfreezing layers and training with standard C.E. Loss.

3.2 Replace-and-Retrain

3.2.1 Quadratic Imitation Learning (QuaIL).

Key Idea: The escaping activation problem encountered when using the polynomial regression strategy was directly caused by the compounding use of polynomials in deeper networks. Specifically, after each pass through the polynomial activation, the output intermediate representation values began to grow exponentially. Following several layers (several passes through the polynomial activations), the intermediate representation values escaped the well-behaving regions of the polynomials and resulted in exploding values (NaNs). The escaping nature of intermediate representations suggests to elevate from simple drop-and-replace strategies to replace-and-retrain strategies which mitigate the escaping activation problem. Rather than attempting to train a network with polynomial activations end-to-end by minimizing the loss between ground truth and predictions, Quadratic Imitation Learning (QuaIL) iteratively builds and trains a neural network with polynomial activations by mimicking the intermediate representation values of a trained ReLU network. Similar to dynamic programming, QuaIL attempts to first solve a sub-problem by mimicking intermediate representation values of a well-behaving network before adding additional layers to a network using polynomial activations. In QuaIL, the polynomial activation function is set to \( f(x) = x^2 \) (Quad).

Setup: Figure 1 depicts the QuaIL training process. First, a ReLU baseline network is trained using standard supervised learning techniques (Fig. 1.1). Then, the first layer of the ReLU network is duplicated and the layer’s ReLUs are replaced with Quad. Here, the Quad network is trained by minimizing the Mean Square Error (M.S.E.) between the first-layer intermediate representations of both networks (Fig. 1.2). In this way, the single-layer Quad network learns to predict similar first-layer representations as the ReLU network. After training converges to a low M.S.E. between the two intermediate representations, the Quad network’s first-layer weights are frozen and the second layer of the ReLU network is cloned and stacked onto the Quad network. Again, ReLU is replaced by Quad for the second layer. Similar to the first layer, the Quad network now minimizes the M.S.E. between the second-layer representations of both networks. This process is repeated until the final layer of the ReLU network has been added to the Quad network and the error between the final representations is minimized (Fig. 1.3).

At this stage, the Quad network is trained using standard supervising learning while gradually unfreezing shallower layers. In the image classification setting, the Cross Entropy (C.E.) loss is minimized between ground truth labels and predictions (Fig. 1.4-1.5).

Result: QuaIL further extends our progress of building deep, P1-friendly networks to AlexNet and VGG11 on the CIFAR-10, CIFAR-100, and TinyImageNet datasets. Each of these networks that only uses the Quad activation function are built up iteratively to limit the effect of escaping activations. However, QuaIL fails to generalize to even deeper networks as even a small difference in the intermediate representations at earlier stages of the deeper networks propagate forward leading to escaping activations and causing training to diverge (denoted as \( \rightarrow \) in Table 2).

Takeaway: QuaIL allows us to iteratively build deep (up to 11-layer) networks with only the Quad activation function but fails to
mitigate unstable intermediate representations for even deeper networks and thus still suffers from escaping activations. For example, a ResNet18 network trained using the QuaIL setup experiences exploding gradients due to escaping activations in latter intermediate representations and is unable to converge during training.

### 3.2.2 Approximate MinMax Normalization.

**Key Idea:** The escaping activation problem encountered during QuaIL illustrates the need to bound pre-activation values to train networks using low-degree polynomial activation functions, especially for deeper neural networks. To do this we developed Approximate Min-Max Normalization (ApproxMinMaxNorm), which places upper and lower constraints on pre-activation values during training by performing a dimension-wise Min-Max normalization:

\[
\tilde{x} = x - \frac{\min x - \frac{x - \min x}{\max x - \min x}}{\alpha} + \beta
\]

where \(\alpha\) and \(\beta\) are scaling parameters. During the training phase, approximations of minimums and maximums are calculated and stored using a weighted moving average of the true minimums and maximums (we use a smoothing factor of 1/10). When performing inference, these stored approximations are then used to perform approximate normalization.

**Setup:** ApproxMinMaxNorm is combined with the QuaIL training procedure; when building the Quad network, ReLU is replaced by an ApproxMinMaxNorm layer immediately followed by QuaIL.

**Result:** We observe stable training for all networks and datasets using QuaIL+AMM. However, at inference time, when using the approximated values of the minimums and maximums for each layer, we again detect the escaping activation problem, albeit to a less degree when compared to the drop-and-replace polynomial regression strategy.

**Takeaway:** ApproxMinMaxNorm prevents the escaping activation problem at training time by explicitly bounding the pre-activation values to polynomial activations. However, the escaping activation problem returns during inference due to approximate minimum and maximum calculations. Thus a true maximum function is required at test time to guarantee bounds on pre-activation values.

### 4 DISCUSSION

The desirable properties of PI-friendly ReLU substitutions are: low multiplicative depth, stability over a sufficiently large range of activation values, and competitive performance when compared to networks with ReLUs. The Quad activation function has been considered a promising solution as its multiplicative depth is one and exhibits stability for simple models and datasets, e.g., MNIST [16]. However, for deeper networks and larger datasets, the desired stability range of pre-activation values increases significantly [30] and using Quad in this extended range results in imprecise approximations of ReLU and poor accuracy [5]. Consequently, higher-degree polynomials are used for more accurate approximation, but suffer from a higher multiplicative depth that results in additional computation (in SS/HE) and noise growth (HE), thus limiting their efficacy in practical settings.

To help mitigate these issues, we devised QuaIL where each layer in the Quad-network learns to mimic intermediate representations of a trained, all-ReLU network. QuaIL worked well for AlexNet and VGG11, which polynomial regression under-performed; however, it did not scale to even deeper networks. To understand why, we dug deeper and found the issue still to be escaping activations. That is, some intermediate representation values still began to grow unbounded. To mitigate escaping activations at training time, we bounded the pre-activation values inputs using an ApproxMinMax normalization strategy, which achieved reasonable accuracies for all the networks except MobileNetV1 and ResNet32. However since the maximum and minimum values were approximated at inference time, the approximation error grew in deeper layers and some activations began to explode (shown in Figure 2 as Quail+AMM). For a better understanding, we replaced the approximated min and max with the true min and max during inference (termed as Quail+MM in Figure 2) and observed that the intermediate representation values were now similar to that of the all-ReLU baseline networks.

Fundamentally, the efficacy of using low-degree polynomials for deeper networks on complex datasets boils down to bounding input values to the polynomial activations in order to mitigate escaping activations, which requires using exact calculations of both the minimum and maximum. However, the issue of calculating exact minimums and maximums brings us back full circle to the problem we were trying to solve: remove all ReLUs (which is defined using

### Table 2: Accuracy when all ReLUs in the networks are substituted with Taylor series approximation (T-Approx.), polynomial regression approximation (Poly-Reg.), QuaIL, and QuaIL+AMM on CIFAR-10/100 (C-10/100) and TinyImageNet (Tiny) datasets.

| Dataset-Net   | Baseline | T-Approx. | Poly-Reg. | QuaIL | QuaIL+AMM |
|---------------|----------|-----------|-----------|-------|-----------|
| MNIST-MLP     | 97.98    | 86.28     | 97.88     | 98.20 | 98.17     |
| MNIST-LeNet   | 99.32    | 9.81      | 99.14     | 99.45 | 99.26     |
| C10-AlexNet   | 85.47    | 12.90     | 1.28       | 71.11 | 79.94     | 79.32     |
| C10-VGG11     | 90.46    | 13.68     | 8.49       | 82.83 | 82.19     | 82.85     |
| C10-VGG16     | 92.78    | 16.01     | 13.31      | 87.57 | —         | 82.25 (82.63) |
| C10-ResNet18  | 93.21    | 1.00      | 13.64      | —     | 83.61 (85.72) |
| C10-MobileNetV1 | 91.70   | 11.04     | 11.13      | 10.00 | 48.03 (49.45) |
| C10-ResNet32  | 91.72    | 26.96     | 90.48      | (90.62) | —         | 56.93 (71.81) |
| C100-AlexNet  | 60.98    | 1.73      | 21.51 (31.8) | 54.83 | 50.76     |
| C100-VGG11    | 68.36    | 1.88      | 3.74       | 52.08 | 55.53     |
| C100-VGG16    | 71.44    | 1.74      | 0.94 (52.83) | —     | 54.56 (55.03) |
| C100-ResNet18 | 74.39    | 1.00      | 1.00       | —     | 65.17 (66.30) |
| C100-MobileNetV1 | 65.00 | 1.00      | 25.8       | —     | 69.20 (61.09) |
| C100-ResNet32 | 67.83    | 4.28      | 66.31      | —     | 19.86 (28.58) |
| Tiny-AlexNet  | 51.65    | 0.51      | 1.70 (8.98) | 38.95 | 36.24     |
| Tiny-VGG11    | 55.36    | 0.66      | 0.5        | 37.59 | 44.63 (44.68) |
| Tiny-VGG16    | 58.88    | 0.45      | 0.56 (3.07) | —     | 45.76 (46.47) |
| Tiny-ResNet18 | 61.59    | 0.50      | 0.5        | —     | 49.45 (53.82) |
| Tiny-MobileNetV1 | 56.16  | 0.76      | 0.46       | —     | 13.43 (22.73) |
| Tiny-ResNet32 | 54.77    | 2.04      | 47.55      | —     | 7.16 (10.60) |

**Figure 2:** Comparison of maximum forward activation values after each nonlinear layer at inference time using Quail+AMM (ApproxMinMax), Quail+MM (TrueMinMax), and an all-ReLU baseline ResNet18.
maximum) to prevent the usage of garbled circuits in PI. We hope the insights gained from Sisyphus aid the PPML community in being mindful when using low-degree polynomial activations in PI-friendly networks.

ACKNOWLEDGEMENTS

This work was supported in part by the Applications Driving Architectures (ADA) Research Center, a JUMP Center co-sponsored by SRC and DARPA. This research was also developed with funding from the Defense Advanced Research Projects Agency (DARPA), under the Data Protection in Virtual Environments (DPRIVE) program, contract HR0011-21-9-0003. The views, opinions and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 308–318.

[2] Ahmad Al Badawi, Jun Chao, Jie Lin, Chan Fook Mun, Jun Jie Sim, Benjamin Hong Meng Tan, Xiao Nan, Khin Mi Mi Aung, and Vijay Ramaseshan Chandrasekar. 2020. Towards the AlexNet Moment for Homomorphic Encryption: HCNN, the First Homomorphic CNN on Encrypted Data with GPUs. IEEE Transactions on Emerging Topics in Computing (2020).

[3] Maxmilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. 2020. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. In Advances in Neural Information Processing Systems 33.

[4] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. 2020. Efficient Monte-Carlo Bayesian Optimization. In 27th USENIX Security Symposium (USENIX Security 18). 1651–1669.

[5] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. 2017. Self-normalizing neural networks. In Proceedings of the 31st international conference on neural information processing systems. 972–981.

[6] Jian Liu, Mika Juuti, Yao Lu, and N Asokan. 2017. Oblivious neural network predictions via minnorm transformations. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 639–651.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition. 770–778.

[8] Tien Duy Tran and Panagiotis G. Karras. 2018. SGN: A Scalable Privacy-Preserving Machine Learning System. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. 639–651.

[9] Parnian Mohassel and Yupeng Zhang. 2017. SecureML: A framework for privacy-preserving machine learning. In 2017 IEEE Symposium on Security and Privacy (SP). 19–38.

[10] Alejandro Molina, Patrick Schrammowski, and Kristian Kersting. 2020. Padio Activation Units: End-to-end Learning of Flexible Activation Functions in Deep Networks. In International Conference on Learning Representations.
[44] M Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based Oblivious Deep Neural Network Inference. In 28th USENIX Security Symposium (USENIX Security 19). 1501–1518.

[45] M Sadegh Riazi, Christian Weinert, Oleksandr Tkachenko, Ebrahim M Songhori, Thomas Schneider, and Farinaz Koushanfar. 2018. Chameleon: A hybrid secure computation framework for machine learning applications. In Proceedings of the 2018 on Asia Conference on Computer and Communications Security. ACM, 707–721.

[46] Bita Darvish Rouhani, M Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: Scalable provably-secure deep learning. In Proceedings of the 55th Annual Design Automation Conference. 2.

[47] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. 2018. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In International Conference on Machine Learning. PMLR, 4490–4499.

[48] Adi Shamir 1979. How to share a secret. Commun. ACM 22, 11 (1979), 612–613.

[49] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. 2016. Understanding and improving convolutional neural networks via concatenated rectified linear units. In International conference on machine learning. PMLR, 2217–2225.

[50] Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014).

[51] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. (2012), 2951–2959.

[52] Patricia Thaine, Sergey Gorbunov, and Gerald Penn. 2019. Efficient evaluation of activation functions over encrypted data. In 2019 IEEE Security and Privacy Workshops (SPW). IEEE, 57–63.

[53] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. arXiv preprint arXiv:1806.03287 (2018).

[54] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853 (2015).

[55] Leon Yao and John Miller. 2015. Tiny imagenet classification with convolutional neural networks. CS 231N 2; 5 (2015), 8.