Privacy-Preserving Deep Models for Plant Stress Phenotyping

Minsu Cho,[1] Koushik Nagasubramanian,[2] Arti Singh,[2] Asheesh Singh,[2]
Soumik Sarkar,[2] Baskar Ganapathysubramanian,[2] Chinmay Hegde[1]

[1] New York University [2] Iowa State University

Abstract

Deep neural networks are increasing being deployed for automating plant stress identification and quantification. However, as this area grows in importance, alleviating privacy concerns of practitioners becomes a major challenge. In this paper, we present a deep learning framework for plant stress phenotyping that *guarantees privacy* to both the owner of the data, as well as the developer of the model. Our framework leverages recent advances in deep neural network design for accelerated private inference (PI) using secure multiparty communication. We showcase our framework on a large-scale image dataset, demonstrate that our newly designed models enjoy nearly 2 orders of magnitude speedup in inference time over state-of-the-art baselines.

Introduction

Plant diseases negatively impact yield potential of crops worldwide, reducing the average annual yield, for instance of soybean by an estimated 11% in the United States. However, disease scouting and phenotyping techniques rely on human scouts and visual ratings. Deep learning promises to automate this labor intensive process by leveraging the availability of massive image datasets acquired in the field. Specifically, these datasets can be used to train deep neural network models that can accurately predict the presence/absence and nature of disease stress in crops from digital image data alone. This promise has begun to be realized in the community via a series of exciting recent works (Pound et al. 2017; Singh et al. 2016; Ghosal et al. 2018).

However, despite these advances, this research direction faces a looming challenge. In order for neural network-based phenotyping to be widely accepted by agricultural practitioners, the twin issues of *privacy* and *trustworthiness* need to be addressed. Typically, image data collection is done by individual field users, while model building and digital phenotyping are done in the data processing divisions of ag-tech companies. However, field users may not wish to entrust their data to a service provider. At the same time, ag-tech companies may not want to share their deep neural network models (potentially trained at enormous financial and human capital expense), with individual users.

Private inference (PI) provides a solution to this challenge. The high level idea is to *guarantee both user and model privacy* using cryptographic techniques (Juvekar, Vaikuntanathan, and Chandrakasan 2018; Mishra et al. 2020a). A typical private inference pipeline involves secure multi-party computation (MPC). At a high level, the goal is to prevent both users and companies from learning anything about each other's parameters. PI techniques thus far reported in the literature have leveraged a range of cryptographic protocols, including homomorphic encryption (HE), additive secret sharing (SS), and garbled circuits (GC). However, a major barrier to widespread deployment of PI is that these all incur heavy computational overheads, resulting in several orders-of-magnitude increase in inference latency compared to standard "plaintext" inference.

The reason for this severe overhead is somewhat curious: *non-linear* operations in deep models, like the well-known Rectified Linear Unit (ReLU), turn out to be the key bottlenecks. For example, in DELPHI (Mishra et al. 2020a), a state-of-the-art PI protocol for deep network inference, it is known that ReLUs account for 93% of ResNet32's online runtime (Jha et al. 2021). This is in direct contrast to standard (plaintext) inference where ReLUs are effectively free, and the dominant runtime costs are due to the floating-point operations of convolutional and fully-connected layers.

In this short paper, we design several novel deep neural network architectures for plant stress phenotyping, specially geared towards preserving privacy of both the user and service provider. The key distinction from standard deep architectures (e.g. VGG, or ResNets) is that our models have a minimal number of nonlinear operations (such as ReLUs) but still achieve near state-of-the-art accuracy. Because our models are ReLU-efficient, efficient private inference using cryptographic protocols can be achieved. Our framework leverages SPHYNX, a new algorithm for designing ReLU-efficient convolutional cells that maximize overall network accuracy under a ReLU budget (Cho et al. 2021).

We demonstrate the effectiveness of SPHYNX using a large-scale image dataset of approximately 16,000 image samples of plant stress (Nagasubramanian et al. 2020). Baseline deep network models for this dataset have been demonstrated to achieve over 90% accuracy, but *each* private inference through these deep models may take hundreds of seconds. Our new models exhibit nearly SOTA accuracy, yet can perform private inference in less than 2 seconds. To the best of our knowledge, these constitute the first results for privacy-preserving models for plant stress phenotyping.

Background on Private Inference

Consider a two-party communication system where the goal is to perform inference (say, classification) on the server's deep neural network model with the client's input data. The client wishes to keep their data private, while simultaneously the server desires to keep their model parameters secret from the client. Private inference techniques set up a protocol for back-and-forth communication such that the server learns nothing about the client's input, the clients learn nothing about the server's model, and in the end both parties are able to access the results of the inference. We adopt a common threat model from prior work in the private inference (Liu et al. 2017; Juvekar, Vaikuntanathan, and Chandrakasan 2018; Mishra et al. 2020b) where both participants are *honest-but-curious*. At a high level, both parties follow the protocol faithfully, but within the protocol may try to infer information about the other party's input/model from the protocol's transcripts.

The SPHYNX framework focuses on the DELPHI cryptographic protocol (Mishra et al. 2020b) for private inference. DELPHI builds on three cryptographic primitives: secret sharing (SS) and homomorphic encryption (HE), for linear operations, and garbled circuits (GC) for non-linear operations. We begin by briefly reviewing these cryptographic primitives.

Additive Secret Sharing (Shamir 1979) allows two parties to hold *additive* shares $[x]_1$, $[x]_2$ of a secret value $x \in \mathbb{F}_p$ (defined over a finite field $\mathbb{F}_p$ where p is a large prime) such that $[x]_1 + [x]_2 = x$. These additive shares can be generated by sampling a random element $r \in \mathbb{F}_p$ and setting $[x]_1 = r$ and $[x]_2 = x - r$.

Homomorphic Encryption (Gentry and Halevi 2011) enables operations on encrypted values without a private key or decryption. A cryptosystem supports a homomorphic operation $(*)$ if for public key (pk), secret key (sk), and ciphertexts $c_1 = Enc(pk, m_1)$, $c_2 = Enc(pk, m_2)$, there exists a function EVAL such that $Dec(sk, \text{EVAL}(c_1, c_2)) = m_1 * m_2$.

Garbled Circuits (Yao 1986) is a scheme introduced by Yao (Yao 1986) that allows two parties to compute a Boolean function f on their private inputs without revealing their inputs to each other. The function f is first represented as a Boolean circuit of two-input logic gates. One of the parties (the garbler) encodes (garbles) the circuit by encrypting the truth table of each gate in the circuit and sends the resulting "garbled circuit" to the other party (the evaluator). The evaluator, computes (or decrypts) the circuit gate-by-gate using encodings of the garbler's inputs and her own inputs, producing an encoding of the circuit's output. She shares this encoding with the garbler, who then reveals the corresponding plaintext.

The Delphi Protocol (Mishra et al. 2020a) is a hybrid protocol that combines the above cryptographic primitives for performing inference with networks with linear and ReLU layers. A description of the protocol is shown in Figure 1,

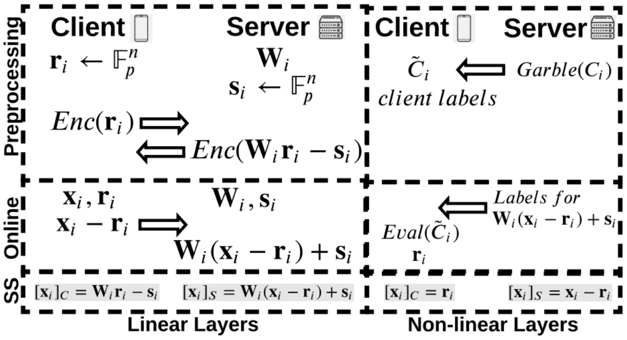

Preprocessing
Online
SS
Client
Server
Client
Server
$\mathbf{r}_i \leftarrow \mathbb{F}_p^n$
$\mathbf{W}_i$
$\mathbf{s}_i \leftarrow \mathbb{F}_p^n$
$Enc(\mathbf{r}_i)$
$Enc(\mathbf{W}_i \mathbf{r}_i - \mathbf{s}_i)$
$\tilde{C}_i$
$Garble(C_i)$
client labels
$\mathbf{x}_i, \mathbf{r}_i$
$\mathbf{W}_i, \mathbf{s}_i$
$\mathbf{x}_i - \mathbf{r}_i$
$\mathbf{W}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$
Labels for
$\mathbf{W}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$
$Eval(\tilde{C}_i)$
$\mathbf{r}_i$
$[\mathbf{x}_i]_C = \mathbf{W}_i \mathbf{r}_i - \mathbf{s}_i$
$[\mathbf{x}_i]_S = \mathbf{W}_i(\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$
$[\mathbf{x}_i]_C = \mathbf{r}_i$
$[\mathbf{x}_i]_S = \mathbf{x}_i - \mathbf{r}_i$
Linear Layers
Non-linear Layers

Figure 1: DELPHI *protocol illustration (Cho et al. 2021):* DELPHI *leverages HE (preprocessing) and SS (online) for the linear layers, and GC for ReLU layers. Note that preprocessing computations are independent to clients' inputs and can be precomputed. Online computation time for linear layers is approximately the same as the plaintext computation; ReLU operations involve expensive online cryptographic computations from GCs, which form the major computational bottleneck.*

DELPHI proceeds in two stages: the preprocessing phase and the online phase. During the preprocessing phase, the client and the server precomputes input independent data that can be used in the online phase. We go over the precomputation procedures for the linear layers and non-linear layers: (1) the client obtains a public key pk and a private key sk from a randomized algorithm; (2) considering the server's model with L layers where W_i is the i$^{\text{th}}$ layers parameters and $i \in \{0, \ldots, L-1\}$, the client and the server samples random vectors $\mathbf{r}_i \in \mathbb{F}_p^n$ and $\mathbf{s}_i \in \mathbb{F}_p^n$, respectively; (3) the client encrypts $\mathbf{r}_i$with public key pk and sends $Enc(pk, \mathbf{r}_i)$ to the server; (4) the server computes $Enc(pk, \mathbf{W}_i\mathbf{r}_i - \mathbf{s}_i)$ homomorphically, and send it back to the client; (4) the client decrypts this ciphertext using the secret key sk and obtains $\mathbf{W}_i\mathbf{r}_i - \mathbf{s}_i$; (5) the server garbles the circuit $\tilde{C}$ for each ReLU in the network and send it to the client, along with labels corresponding to $\mathbf{r}_{i+1}$ and $\mathbf{W}_i\mathbf{r}_i - \mathbf{s}_i$.

During the online phase, DELPHI leverages the different cryptographic primitives depending on linear operations or non-linear operations. In linear layers, the client computes and sends $\mathbf{x}_i - \mathbf{r}_i$ to the server. The client and server now hold secret shares of the client's input $\mathbf{x}_i$, or equivalently, the first layer's inputs. The server then computes $\mathbf{W}_i \cdot (\mathbf{x}_i - \mathbf{r}_i) + \mathbf{s}_i$, its own share of the i$^{\text{th}}$ layer's output. In non-linear layers, especially ReLU, the server sends the encoding of $\mathbf{W_i}.(\mathbf{x_i} - \mathbf{r_i}) + \mathbf{s_i}$ to the client, who is now able to evaluate the garbled circuit and send the encoded output to the server. The server decodes the garbled circuit output, which is $\mathbf{x_{i+1}} - \mathbf{r_{i+1}}$, the server's share for the next linear layer.

From the above discussion, we can observe that during the online phase of private inference, computation involving linear layers in the network are effectively the same cost as standard (plaintext) computation. On the other hand, *operations involving ReLU (or max-pool) layers use garbled circuits that require expensive online crpytographic computations and interaction between parties, resulting in high latency*. This motivates the need for neural architectures that judiciously minimize ReLU computations. We next describe a recent framework, SPHYNX, which serves this need.

The SPHYNX Framework

To design new ReLU-efficient networks, we will use SPHYNX, a new network design approach presented in (Cho et al. 2021). SPHYNX is a *micro-search* technique, i.e., it discovers *cell architectures* that can be repeated several times to form deep networks. As is typical for micro-search methods, there are two types of cells being designed: *normal* cells that preserve feature map size, and *reduce* cells that decrease feature resolution. At a high level, SPHYNX consists of two main innovations: (i) a new ReLU-efficient cell search space, which can be combined algorithmic with any existing neural architecture search (NAS) methods to discover promising cell architectures; (ii) a new stochastic optimization method to discover optimal the locations of *reduce* cells in terms of layer depth.

SPHYNX: Search Space

SPHYNX ues a new ReLU-efficient search space which will influence later search strategies. We start with the traditional DARTS search space (Liu, Simonyan, and Yang 2018), which can be viewed as a multigraph with nodes (representing feature maps) and edges (representing operations such as different types of convolution, pooling, and so on). Traditional NAS techniques aim to select the best sub-graph that maximizes test accuracy.

To this search space, we make a few changes. First, we eliminate the ReLU layer from convolution operations so each ReLU-Conv-BN sequence is now simply a Conv-BN sequence. We also replace separable convolutions with vanilla convolutions, since they tend to be more expressive. We also remove all max-pooling operations since these are also nonlinear operations that require expensive GCs to compute. Therefore, the only non-linearity in our new SPHYNX search space is a ReLU layer at the output of each cell. In addition, SPHYNX follows the ReLU balancing rule introduced in CryptoNAS (Ghodsi et al. 2020). Unlike conventional FLOP balancing methods which doubles the channel size when the spatial resolution is halved, ReLU balancing quadruples channel size in order to distribute ReLU equally across layers.

SPHYNX: Search Phase

Finding cells. Having defined the search space, we now discover *normal*/*reduce* cells using the DARTS micro-cell search algorithm. The choice of DARTS here is entirely for convenience, and we should emphasize that SPHYNX can be used in conjunction with any other search method such as ENAS (Pham et al. 2018), GDAS (Dong and Yang 2019), PC-DARTS (Xu et al. 2020), and GAEA (Li et al. 2021); the best choice of NAS method is left as future work.

Learning optimal location of reduce cells. If the overall network has D cell layers, conventional cell-based NAS approaches, including DARTS, fix the position of *reduce* cells at $D/3$ and $2D/3$ cell-depth index. In contrast, we propose a method to find the optimal position of *reduce* cells to improve network performance.

We focus on the case with two *reduce* cells; extending this to more than two *reduce* cells is straightforward. Given a network with D cells, let $\boldsymbol{\beta} \in \mathbb{R}^K$ be a position indicator vector where $K = \binom{D}{2}$ is the number of all possible choices of *reduce* cell locations, and let $\hat{\boldsymbol{\beta}} = \text{softmax}(\boldsymbol{\beta}) = \frac{\exp \beta_i}{\sum_k \exp \beta_k}$ be a probability distribution over K elements.

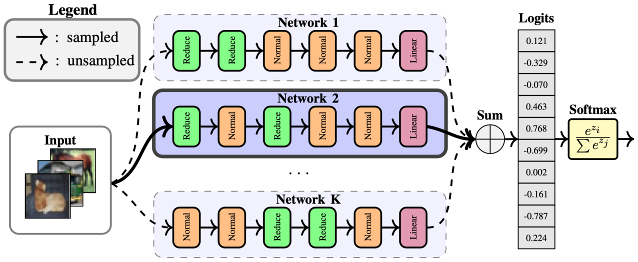


Figure 2: *Illustration of the Gumbel-softmax trick applied to searching reduce cell location. We randomly sample a candidate network from categorical variable $\boldsymbol{\beta}$, and only train the sampled network for a given batch. In this figure, for this iteration our algorithm samples Network 2 and the sampled network weights and categorical variable $\boldsymbol{\beta}$.*

Define a categorical random variable with distribution $\hat{\boldsymbol{\beta}}$ and encoded by random one-hot vectors $\mathbf{g} \in \{0, 1\}^K$. We construct a "super" network as shown in Figure 2. Let f_i, where $i \in \{1, 2, \ldots, K\}$ be a function parameterized by weights $\mathbf{w}_i$; we can imagine f_i to represent candidate networks with different locations of *reduce* cells. The output F computes the linear combination $F = g_1 f_1 + \ldots + g_k f_k$. Intuitively, F samples one branch according to $\mathbf{g}$.

One can imagine learning the optimal indicator vector $\boldsymbol{\beta}$ via gradient descent; unfortunately, the sampling operation is not differentiable. Therefore, we leverage the Gumbel-softmax trick (Jang, Gu, and Poole 2017). During the forward pass, we sample a one-hot vector according to the formula:

$$\mathbf{g} = \text{one-hot}\left(\underset{i \in \{1,2,\ldots,K\}}{\arg\max} \; G_i + \log(\hat{\beta}_i) \right) \tag{1}$$

where $G_i \sim \text{Gumbel}(0, 1)$ i.i.d samples drawn from the standard Gumbel distribution. During the backward pass, we use the straight-through Gumbel softmax estimator which replaces $\mathbf{g}$ with $\tilde{\mathbf{g}}$ during the gradient update:

$$\tilde{g}_i = \frac{\exp\left(\log g_i + G_i\right)/\tau}{\sum_k (\exp\left(\log g_k + G_k\right))/\tau} \tag{2}$$

where τ is a temperature parameter. The parameter τ controls the sharpness of the softmax approximation; $\tilde{\mathbf{g}} = \mathbf{g}$ as $\tau \to 0$, whereas $\tilde{\mathbf{g}}$ becomes an uniform distribution as $\tau \to \infty$. Equipped with a fully differentiable technique for learning the parameter $\boldsymbol{\beta}$, we now train both the network parameters $\mathbf{w}$ and categorical parameter $\boldsymbol{\beta}$.

Once the training terminates, we select the positions of the *reduce* cells in the final network looking at the peak achieved by the categorical distribution (without any Gumbel sampling): $\mathbf{g} = \text{one-hot}(\arg\max_{i \in \{1,2,\ldots K\}} \log(\hat{\beta}_i))$.

Experimental results

We now show the effectiveness of SPHYNX for designing ReLU-efficient networks for plant stress phenotyping.

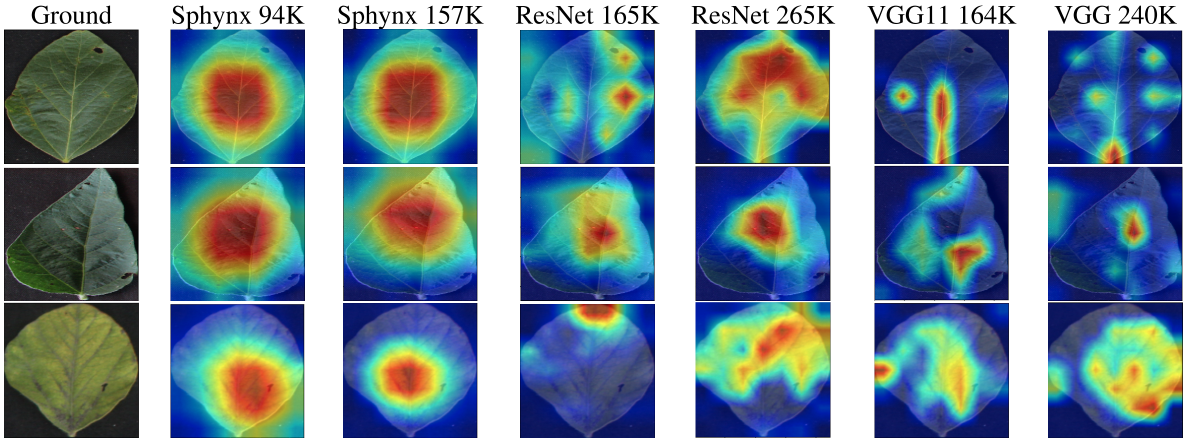


Figure 3: GradCaM outputs for various models. Each row's class from top to bottom corresponds to **Bacterial Blight**, **Frogeye Leaf Spot**, and **Iron Deficiency Chlorosis**, respectively.

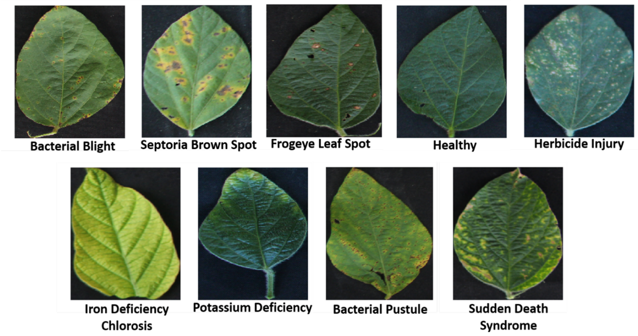


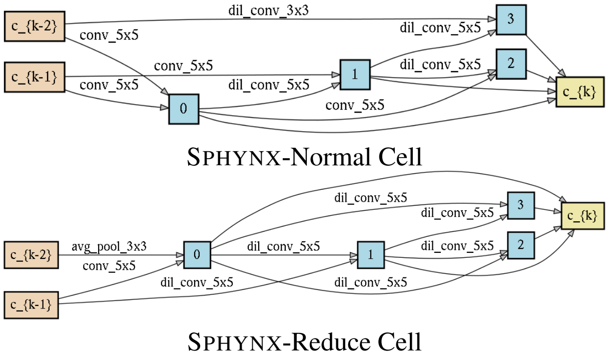
Figure 4: *Sample images from each class in our dataset. These classes cover a diverse spectrum of biotic and abiotic foliar stresses.*

Dataset. We report results on a large-scale image dataset consisting of 16,573 RGB images of soybean [Glycine max L. (Merr.)] leaves across nine different classes (i.e., eight different soybean stresses, and the ninth class containing healthy soybean leaves). More details on the imaging setup and dataset collection can be found in (Ghosal et al. 2018). Briefly, these classes cover a diverse spectrum of biotic and abiotic foliar stresses in soybean. Figure 4 illustrates the nine different soybean leaf classes used in this study.

Results Table 1 shows accuracy and online runtime values of several networks obtained using our approach. For comparisons, we show two baseline models (VGG-11 and ResNet-18) which achieve SOTA accuracy of $>$ 93% on our dataset. However, note that these models are extremely ReLU heavy, and therefore private inference on these models incurs a latency of hundreds of seconds *per test image*, which is far too slow in terms of user experience. Simple channel- and feature-map scaling of the original models can be performed to reduce the latency to 3-5 seconds, but with significant accuracy drop.

Table 1: Test accuracy and PI latency results.

Architecture	ReLUs	Test Acc.	PI Online Lat.
SPHYNX	94K	**87.73**%	**2.012**s
SPHYNX	157K	89.29%	3.344s
VGG11	164K	80.14%	3.451s
VGG11	240K	88.75%	5.045s
VGG11 (original)	9642K	93.98%	202.49s
ResNet18	165K	78.58%	3.472s
ResNet18	265K	86.64%	5.575s
ResNet18 (Original)	6523K	93.32%	136.47s



SPHYNX-Normal Cell

SPHYNX-Reduce Cell

Figure 5: *Normal* and *reduce* cells found by SPHYNX.

In contrast, the networks designed by SPHYNX demonstrate an accuracy of nearly 90%, but only require approximately 100K ReLUs, and a runtime of 2-3 seconds. This represents *2 orders of magnitude* speedup in online private inference runtime, and our networks conceivably can be deployed in practice. Figure 5 displays the structure of the *normal* and *reduce* cells found using our algorithm.

In Figure 3 we also display interpretability maps achieved using the different models. We observe that while the models from SPHYNX achieve excellent test accuracy, the interpretability maps appear quite different from baseline methods. Understanding the role of architecture for interpreting the functioning of various models is left for future work.

Acknowledgements

This work was supported in part by the National Science Foundation (under grants CCF-2005804 and CNS-1954556) and the USDA/NIFA (under grant 2021-67021-35329).

References

Cho, M.; Ghodsi, Z.; Reagen, B.; Garg, S.; and Hegde, C. 2021. Sphynx: ReLU-Efficient Network Design for Private Inference. *arXiv preprint arXiv:2106.11755*.

Dong, X.; and Yang, Y. 2019. Searching for A Robust Neural Architecture in Four GPU Hours. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1761–1770.

Gentry, C.; and Halevi, S. 2011. Implementing gentry's fully-homomorphic encryption scheme. In *Annual international conference on the theory and applications of cryptographic techniques*, 129–148. Springer.

Ghodsi, Z.; Veldanda, A. K.; Reagen, B.; and Garg, S. 2020. CryptoNAS: Private Inference on a ReLU Budget. In Larochelle, H.; Ranzato, M.; Hadsell, R.; Balcan, M. F.; and Lin, H., eds., *Advances in Neural Information Processing Systems*, volume 33, 16961–16971. Curran Associates, Inc.

Ghosal, S.; Blystone, D.; Singh, A. K.; Ganapathysubramanian, B.; Singh, A.; and Sarkar, S. 2018. An explainable deep machine vision framework for plant stress phenotyping. *Proceedings of the National Academy of Sciences*, 115(18): 4613–4618.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical reparameterization with gumbel-softmax.

Jha, N. K.; Ghodsi, Z.; Garg, S.; and Reagen, B. 2021. DeepReDuce: ReLU Reduction for Fast Private Inference. In *International Conference on Machine Learning*.

Juvekar, C.; Vaikuntanathan, V.; and Chandrakasan, A. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 18)*, 1651–1669.

Li, L.; Khodak, M.; Balcan, N.; and Talwalkar, A. 2021. Geometry-Aware Gradient Algorithms for Neural Architecture Search. In *International Conference on Learning Representations*.

Liu, H.; Simonyan, K.; and Yang, Y. 2018. Darts: Differentiable architecture search. *Proc. Int. Conf. Machine Learning*.

Liu, J.; Juuti, M.; Lu, Y.; and Asokan, N. 2017. Oblivious neural network predictions via minionn transformations. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 619–631.

Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; and Popa, R. A. 2020a. Delphi: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*, 2505–2522. USENIX Association. ISBN 978-1-939133-17-5.

Mishra, P.; Lehmkuhl, R.; Srinivasan, A.; Zheng, W.; and Popa, R. A. 2020b. DELPHI: A Cryptographic Inference Service for Neural Networks. In *29th USENIX Security Symposium (USENIX Security 20)*.

Nagasubramanian, K.; Jubery, T. Z.; Ardakani, F. F.; Mirnezami, S. V.; Singh, A. K.; Singh, A.; Sarkar, S.; and Ganapathysubramanian, B. 2020. How useful is active learning for image-based plant phenotyping? *arXiv preprint arXiv:2006.04255*.

Pham, H.; Guan, M. Y.; Zoph, B.; Le, Q. V.; and Dean, J. 2018. Efficient neural architecture search via parameter sharing. *Proc. Int. Conf. Machine Learning*.

Pound, M. P.; Atkinson, J. A.; Townsend, A. J.; Wilson, M. H.; Griffiths, M.; Jackson, A. S.; Bulat, A.; Tzimiropoulos, G.; Wells, D. M.; Murchie, E. H.; et al. 2017. Deep machine learning provides state-of-the-art performance in image-based plant phenotyping. *Gigascience*, 6(10): gix083.

Shamir, A. 1979. How to share a secret. *Communications of the ACM*, 22(11): 612–613.

Singh, A.; Ganapathysubramanian, B.; Singh, A. K.; and Sarkar, S. 2016. Machine learning for high-throughput stress phenotyping in plants. *Trends in plant science*, 21(2): 110–124.

Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.-J.; Tian, Q.; and Xiong, H. 2020. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *International Conference on Learning Representations*.

Yao, A. C.-C. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*, 162–167. IEEE.