Reducing ReLU Count for Privacy-Preserving CNN Speedup

Inbar Helbitz,[1] **Shai Avidan** [2]

[1] School of CS, Tel-Aviv University
[2] School of EE, Tel-Aviv University
inbarhelbitz@mail.tau.ac.il, avidan@eng.tau.ac.il

Abstract

Privacy-Preserving Machine Learning algorithms must balance classification accuracy with data privacy. This can be done using a combination of cryptographic and machine learning tools such as Convolutional Neural Networks (CNN). CNNs typically consist of two types of operations: a convolutional or linear layer, followed by a non-linear function such as ReLU. Each of these types can be implemented efficiently using a different cryptographic tool. But these tools require different representations and switching between them is time-consuming and expensive. Recent research suggests that ReLU is responsible for most of the communication bandwidth.

ReLU is usually applied at each pixel (or activation) location, which is quite expensive. We propose to share ReLU operations. Specifically, the ReLU decision of one activation can be used by others, and we explore different ways to group activations and different ways to determine the ReLU for such a group of activations.

Experiments on several datasets reveal that we can cut the number of ReLU operations by up to three orders of magnitude and, as a result, cut the communication bandwidth by more than 50%.

Introduction

Deep Learning algorithms have made great progress in recent years. Therein lies the problem. In order to classify an image, a Deep Learning algorithm must be exposed to its content. Yet, an image contains private information that should not be exposed. Protecting the privacy of the data is paramount in the medical, financial, and military domains, to name a few.

Privacy-Preserving Machine Learning algorithms are designed to address this problem. Such an algorithm will protect the privacy of the data, as well as that of the network, while performing the task at hand. For example, assume that Alice owns an image and Bob owns a Convolutional Neural Network (CNN) for image classification. Their goal is to collaborate in classifying the image without revealing any additional information. Alice should learn nothing about the weights of Bob's network, while Bob should learn nothing about the content of Alice's image. Only the classification result should be revealed to the agreed upon party.

This problem can be solved using cryptographic tools. The leading approach in recent work is to use secure Multi-Party Computation (MPC) techniques. In this setting, Alice and Bob securely evaluate the network layer by layer until an outcome is reached. At each stage, Alice and Bob hold private shares of the data (or the intermediate results of the network so far). Then, they use Homomorphic Encryption to compute the convolutional and linear component. The non-linear ReLU operation requires a different representation and so Alice and Bob must switch their data representation to compute the ReLU outcome of their operation. And the process repeats for the next layer. Switching to and from ReLU representation is time-consuming and requires considerable communication bandwidth.

The goal of this paper is to develop an algorithm that reduces the ReLU count in a given CNN. Our solution is based on *ReLU Sharing* - a group of pixels (or activation units down the network) operate based on a single ReLU operation. Today, each activation performs the ReLU operation on the outcome of the convolution at the activation's site. In ReLU sharing, the ReLU decision at one activation location is shared by others. For example, if we assume that nearby activations are highly correlated, then it is reasonable to assume that their ReLU result will be similar. Therefore, it might be enough to perform ReLU on one activation, and use that result for other activations in its neighborhood.

We consider different methods to group activations and evaluate different ways for a group of activations to perform a shared ReLU operation. We evaluate our algorithm on several data sets and report the results. We find that in some cases we can reduce the ReLU count by up to three orders of magnitude in early layers of the network and cut the total number of ReLU operations by nearly 80% with less than a 2% drop in accuracy.

Background

Secure Multi Party Computation (MPC) was first investigated in Cryptography. Yao (Yao 1982) proposed a Garbled Circuit (GC) to solve the Millionaire Problem: Two Millionaires, Alice and Bob, want to determine which one has more money, without revealing their true wealth. Yao reduced the problem to a circuit that is evaluated jointly and securely by both parties. Observe that the Millionaire Problem is the same as ReLU.

Since every computer program can be reduced to a Boolean circuit, then any computer program can be evaluated securely. Later, Goldreich *et al.* (Goldreich, Micali, and Wigderson 1987) extended MPC to more than two parties. In practice, GC suffers from two main drawbacks. It requires multiple rounds of communication and is extremely slow to compute.

Alternatively, one can use Fully Homomorphic Encryption (FHE), which is an encryption scheme that supports addition and multiplication of encrypted numbers. This allows Alice to encrypt her image and send it to Bob. Bob, in turn, can run his Convolutional Neural Network (CNN) on the encrypted data and return the result. Because FHE does not support a non-linearity such as ReLU, Bob will have to use an alternative non-linearity such as a low degree polynomial or *tanh*.

This elegant solution guarantees the privacy of both parties, while minimizing communication bandwidth. Alice simply sends her encrypted image to Bob and gets in return its classification. This solution was adopted in CryptoNets (Dowlin et al. 2016). Unfortunately, FHE is painfully slow and running it on real images is not feasible currently.

Which leads us to present day solutions. Instead of relying on FHE, most methods today evaluate the CNN layer by layer using a combination of GC and Homomorphic Encryption (HE). These algorithms require several rounds of communication and rely on Homomorphic Encryption (HE), which is less demanding than FHE. Specifically, HE supports either addition or multiplication of encrypted data and is much faster to compute. This means that Bob can run the convolutional layers of his network on Alices's encrypted data, but not the ReLU non-linearity. The ReLU operation can be implemented using a Garbled Circuit. Switching between HE and GC is very time consuming and requires several rounds of communication between Alice and Bob.

Most of the recent progress in Secure Deep Learning mixes and matches these cryptographic tools. DeepSecure (Rouhani, Riazi, and Koushanfar 2018) build on top of Yao's Garbled Circuit a system for secure Deep Learning, and test it on several data sets from different domains. In the context of Vision, they report results on the MNIST dataset.

SecureML (Mohassel and Zhang 2017) use two-party computation to address the problem. In particular, they develop new techniques to support secure arithmetic operations on shared decimal numbers, and propose MPC-friendly alternatives to non-linear functions such as sigmoid and softmax that are superior to prior work.

MiniONN (Liu et al. 2017) transforms a CNN into an Oblivious CNN. They detail how each component in a CNN can be implemented obliviously and show that some of the computational burden can be shifted to an offline stage. They show considerable improvements over both CryptoNets and SecureML. GAZELLE (Juvekar, Vaikuntanathan, and Chandrakasan 2018) further improved the efficiency of the linear layer computation.

Chamelon (Riazi et al. 2018) combines secure function evaluation with secret sharing for the linear layers and MPC or GC for the nonlinear operations. They report results that are two orders of magnitude faster than CryptoNets (Dowlin et al. 2016) and about four times better than (Liu et al. 2017).

Recently, FALCON (Li et al. 2020) suggested the use of FFT to accelerate the secure computation of the linear layer. They are also the first to introduce a secure softmax evaluation.

The work most closely related to us is that of Shafran *et al.* (Shafran et al. 2019). They, too, observe that ReLUs are expensive to compute and propose several ways to reduce their count.

Method

Our goal is to reduce the number of ReLU operations, as this is a major factor in the cost of secure inference. Our key insight is that activations are highly correlated with each other and hence the ReLU decision of one can be used for another. The question is how to share ReLU decisions without compromising privacy?

Let $\mathbf{x} \in R^{n \times m}$ denote an $n \times m$, single channel image, and let $\mathbf{w} \in R^{k \times k}$ denote a convolution kernel. The convolution result is given by $\mathbf{s} = \mathbf{x} * \mathbf{w}$ and the ReLU on top of it is $\mathbf{y} = \text{ReLU}(\mathbf{s})$, which is defined element-wise as:

$$\text{ReLU}(\mathbf{s}(\mathbf{p})) = \begin{cases} \mathbf{s}(\mathbf{p}) & \mathbf{s}(\mathbf{p}) \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

where $\mathbf{p}$ is (2D) activation location.

We can generalize ReLU such that a ReLU decision of a group of activations is then being used by each activation independently. Formally:

$$\text{gReLU}(\mathbf{s}(\mathbf{p}), \mathbf{v}^T\mathbf{s}) = \begin{cases} \mathbf{s}(\mathbf{p}) & \mathbf{v}^T\mathbf{s} \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

where $\mathbf{v}$ is a weight vector, the size of $\mathbf{s}$, and we treat them both as column vector for ease of notation. If, for $\mathbf{s}(\mathbf{p})$, we take $\mathbf{v}$ to be a one hot vector with $\mathbf{v}(\mathbf{p}) = 1$ then we are back to standard ReLU. If, on the other hand, we take $\mathbf{v}$ to be a one hot vector with $\mathbf{v}(\mathbf{q}) = 1$ then the ReLU decision for pixel $\mathbf{p}$ is determined by pixel $\mathbf{q}$.

Of course, $\mathbf{v}$ does not have to be a one hot vector, and we do not have to use just a single vector $\mathbf{v}$. A straightforward generalization is to group the activations in the channel plane into patches, assign a different weight vector $\mathbf{v_i}$ to each patch and learn the weights.

$$\text{gReLU}(\mathbf{s}(\mathbf{p}), \mathbf{v_i}^T\mathbf{s}) = \begin{cases} \mathbf{s}(\mathbf{p}) & \mathbf{v_i}^T\mathbf{s} \geq 0 \land \mathbf{p} \in i \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

where $\mathbf{p} \in i$ is a shorthand for activation $\mathbf{p}$ belongs to patch i.

Figure 1 shows the different ReLU variants that can be used. From left to right we see, first, a regular ReLU, where the ReLU operation is applied directly to the activation itself. Next we show gReLU, where the ReLU decision of activation $\mathbf{q}$ (green activation) is applied to the red-boundary activation $\mathbf{p}$. The last two sub-plots show two different grouping techniques that can be used. In both cases, the

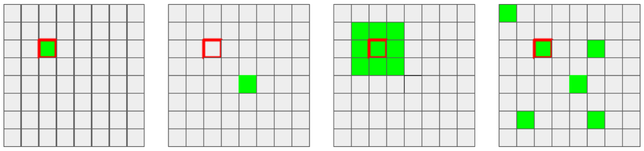

Figure 1: Illustration of ReLU versions. Four toy 8×8 pixel images. The green pixels determine the ReLU of the pixel with red boundary. From left to right: Regular ReLU, gReLU - the ReLU of the red pixel is determined by a different pixel (marked in green), gReLU - the ReLU of the red pixel is determined by all green pixels (grouped based on proximity in image plane), and gReLU - the ReLU of the red pixel is determined by a group of green pixels (clustered based on training data).

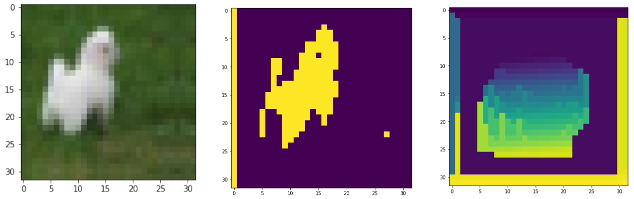

Figure 2: ReLU clustering. (left) An example input image. (Center) ReLU activation map for that image for one of the channels in the first layer. (Right) A pixel clustering map based on the ReLU activation maps of all images in the training set.

gReLU decision of the weighted sum of the green activations is used to determine the ReLU operation of the red-boundary activation.

The first clustering technique is oblivious to data and simply breaks the channel into small patches (i.e., 3×3 patches). The second clustering technique we consider is based on the training data, as we explain next.

Assume we have N training sample and let $\mathbf{y^j}(\mathbf{p})$ be a binary vector that denotes the ReLU decision of activation $\mathbf{p}$ for example j. Let $\mathbf{y}(\mathbf{p}) = (y^1(\mathbf{p}), \cdots, y^N(\mathbf{p}))$ denote an N dimensional vector of the ReLU response, at pixel $\mathbf{p}$ of all training examples. The number of such vectors is the number of activations in the channel, which is nm. We now cluster these vectors to find activations that are correlated with each other.

Figure 2 shows an example taken from the CIFAR-10 dataset. On the left we see an example input image, at the center we see its corresponding ReLU activation map $\mathbf{y^j}$ for one particular channel in the first layer of the network and on the right we see the clustering map. There is a different weight vector $\mathbf{v_j}$ for each color in the clustering map (not shown). The weights are set during training.

One problem with the gReLU operator is that all pixels in the group behave the same. Either they all maintain their input value, or they are all set to zero. If we use a single weight vector per channel, this means that either the entire channel is preserved or zeroed out. To avoid this situation, we introduce a noisy version of gReLU:

$$\text{ngReLU}(\mathbf{s_j}, \mathbf{v}^T\mathbf{s}, p) = \left\{ \begin{array}{ll} \text{gReLU}(\mathbf{s_j}, \mathbf{v}^T\mathbf{s}) & \text{Ber}(p) \\ \text{gReLU}(\mathbf{s_j}, -\mathbf{v}^T\mathbf{s}) & \text{otherwise} \end{array} \right. \tag{4}$$

where Ber(p) is the Bernoulli distribution with parameter p. This way, some activations maintain their value in case the ReLU decision is to zero out. In case the ReLU decision was to keep the activation value, then zero out some of the activations is akin to Dropout, which is a known technique in the literature.

In terms of security, the first clustering method, which is oblivious to the data, preserves the privacy of the data. No additional information is leaked to either party, other than the fact that some activation units share the same ReLU output, which is akin to sharing the overall architecture of the network between the parties. The second clustering method, which is based on the *training* data, reveals information, but only about the training data and *not* about the query image at inference time. See, for example, Figure 2 or Figure 4.

Experiments

Datasets We test our algorithms on the CIFAR-10, SVHN and Fashion-MNIST datasets. They are 10-class classification problems that take as an input an RGB images of size 32×32 pixels (CIFAR-10, SVHN) or a grayscale image of size 28×28 (Fashion-MNIST).

Implementation Details In order to evaluate running times, number of communication rounds and the communication bandwidth in the encrypted settings we used the PySyft library (OpenMined 2018), a framework for secure and private deep learning within the PyTorch framework. PySyft is a 3-party computation settings implementing the secureNN protocol (Wagh, Gupta, and Chandran 2019).

Table 1 gives the round complexity and communication bandwidth of the different operations involved in computing a single layer in a CNN for the secureNN protocol. $Linear_{m,n,v}$ denotes a matrix multiplication of dimension $m \times n$ with $n \times v$. $Conv2d_{m,i,f,o}$ denotes a convolutional layer with input $m \times m$, i input channels, a filter of size $f \times f$, and o output channels. DReLU denotes the binary calculation if a value is greater than 0 for a single value. Mul denotes a single value multiplication. A single ReLU operation is composed of a DReLU and a Mul operations. The round complexity of a convolution layer is 2, while the total round complexity of ReLU layer is 10.

All communication is measured for l-bit inputs (64-bit for PySyft) and p denotes the field size (67 for PySyft), for more details see (Wagh, Gupta, and Chandran 2019). ReLU clearly dominates the process.

We evaluate the run times of our models on a single Intel Core i5 CPU and 8GB RAM, which lets us evaluate improvement in performance. In our setting both the model and the data are shared between 2 parties (Alice and Bob), the third party doesn't hold shares but helps with common randomness and computations. In our setting all 3 parties live on the same machine and do not communicate over the network, so all running times do not consider network latency.

Protocol	Rounds	Communication
$Linear_{m,n,v}$	2	$(2mn + 2nv + mv)l$
$Conv2d_{m,i,f,o}$	2	$(2m^2f^2i + 2f^2oi + m^2o)l$
DReLU	8	$(8logp + 19)l \approx 83l$
Mul	2	$5l$

Table 1: Round and communication complexity for secureNN protocols. Linear and convolutional layers require just 2 round of communication. ReLU operation requires both DReLU and Mul operations, totaling in 10 rounds of communication. See text for further details.

Operation	patch size	Output size
conv	3×3	$32 \times 32 \times 64$
conv	3×3	$32 \times 32 \times 64$
Avg Pool	2×2	$16 \times 16 \times 64$
conv	3×3	$16 \times 16 \times 64$
conv	3×3	$16 \times 16 \times 64$
Avg Pool	2×2	$8 \times 8 \times 64$
conv	3×3	$8 \times 8 \times 64$
conv	1×1	$8 \times 8 \times 64$
conv	1×1	$8 \times 8 \times 16$
FC	1024×10	1×10

Table 2: CIFAR-10 Architecture. The input is a $32 \times 32 \times 3$ RGB image. After each convolution we perform regular ReLU. The first two layers account for about 75% of the ReLU operations in the network.

CIFAR-10 For the CIFAR-10 dataset, we use a CNN that is based on the architecture proposed in MiniONN (Liu et al. 2017). See table 2. The network consists of a sequence of convolutions, ReLU, max pooling, and fully connected layers. We replaced the max pooling operation with average pooling, to reduce ReLU count. The number of ReLU operations in the first and second layers account for about 75% of all ReLU operations, so we focus on them.

SVHN and Fashion-MNIST For the SVHN and the Fashion-MNIST datasets, we use the CNNs in tables 3 and 4, respectively. Since they are small networks we focus on the first ReLU layer of the networks. For SVHN the first layer accounts for 92% of all ReLU operations, and for Fashion-MNIST the first layer accounts for 60%.

Operation	patch size	Output size
conv	4×4	$29 \times 29 \times 32$
Avg Pool	3×3	$9 \times 9 \times 32$
conv	4×4	$6 \times 6 \times 64$
FC	2304×10	1×10

Table 3: SVHN Architecture. The input is a $32 \times 32 \times 3$ RGB image. After each convolution we perform regular ReLU. The first ReLU layer accounts for about 92% of the ReLU operations in the network.

Operation	patch size	Output size
conv	3×3	$28 \times 28 \times 32$
Avg Pool	2×2	$14 \times 14 \times 32$
conv	3×3	$14 \times 14 \times 64$
Avg Pool	2×2	$7 \times 7 \times 64$
Dropout 0.5	$-$	$-$
FC	3136×128	1×128
FC	128×10	1×10

Table 4: Fashion-MNIST Architecture. The input is a 28×28 grayscale image. After each convolution we perform regular ReLU. The first ReLU layer accounts for about 60% of the ReLU operations in the network.

Reducing ReLU Count The first set of experiments is designed to measure the impact of reducing ReLU count on the CNN accuracy. This can be achieved by either clustering activations into small image patches (i.e., data agnostic), or clustering together based on their behaviour on the training data (i.e., data dependent).

We start with sharing ReLUs over small patches of activations (i.e., pixels in the first layer). In particular, we evaluate non-overlapping patches of size 3×3 and 4×4 activations, and perform this on both the first and second layer. For comparison, we also evaluate the two extreme cases. The first is the regular CNN practice (i.e., 1×1 patches), the other is the extreme case where the entire image (i.e., patch size of 32×32) is based on a single ReLU.

In each case we assign a weight vector and set its weights during training. All activations in the patch base their ReLU decision on it. This partitioning is agnostic to the content of the images. The top part of Table 5 shows the results of this experiment on the CIFAR-10 model. Observe the nice correlation between the number of ReLUs and the accuracy.

For the SVHN and Fashion-MNIST model see Table 6. We get less than 1% loss in accuracy, compared with the original accuracy of the models but with only 81 ReLUs per channel, compared to 841 and 784 ReLUs per layer, respectively, in the original models.

The next part of Table 5 reports results of keeping one layer intact and replacing the other layer with a single ReLU per layer. We see that performance actually increases in the case of the first layer but degrades in the case of the second layer. As we go deeper into the network we need more and more ReLU decisions in order to maximize the model's accuracy.

The extreme case of using a single ReLU decision for the whole channel is like a binary choice whether to use this channel in the classification of the image or not, since it either leaves the activation map as is or zeroes-out the entire channel.

As discussed earlier, if the ReLU decision is zero, then the entire patch is zeroed out, leaving no information for the next pooling operation. To battle this phenomenon, we introduce noise into the ReLU result. Specifically, in this experiment both layers use a single ReLU per channel, where the ReLU is carried out on the middle activation of the channel but we add 20% noise to the ReLU decision of the sec-

Method	Patch size	#ReLU	Noise	Accuracy
Original	1×1	$(1024, 1024)$	No	83.59%
Uniform	3×3	$(121, 121)$	No	82.65%
Uniform	4×4	$(64, 64)$	No	81.58%
Uniform	32×32	$(1, 1)$	No	81.13%
Layer 1 only	32×32	$(1, 1024)$	No	84.36%
Layer 2 only	32×32	$(1024, 1)$	No	79.55%
Noise	32×32	$(1, 1)$	20%	80.10%
FC+Noise	32×32	$(1, 1)$	20%	82.01%

Table 5: CIFAR-10 ReLU sharing on image patches. The expression (a, b) in the #ReLU column counts the number of ReLU operations in the first (a) and second (b) per channels. The original network requires $2048 = 1024 + 1024$ ReLU operations per channel with an accuracy of 83.59%. The "Layer 1 only" ("Layer 2 only") replaces all ReLU operations of the first (second) layer with a single ReLU operation. The "FC+Noise" method requires just 1 ReLU operations per channel with an accuracy of 82.01%. A drop of three orders of magnitude in ReLU count. See more details in the text.

ond layer only. Results for CIFAR-10 are reported in Table 5 and for SVHN in Table 6. As can be seen, adding noise helped improve the results compared to working with image patches.

Next we replaced the use of a single activation to determine the ReLU outcome with learned weights of the activations. To do that, we added a fully connected layer and performed a dot-product with the activation map and got a single value that determines the ReLU decision. Then we flipped 20% of the activation decisions and according to this we perform the ReLU on the original activation map. For CIFAR-10, results are shown in table 5. We get less than 1% loss in accuracy, compared with the original accuracy of the model but with only 1 ReLUs per channel per layer, compared to 1024 ReLUs per layer in the original model.

We observe that it is easier to compress the earlier layers of the network and conjecture that this is because the resulting image is smoother. To validate this assumption, we compute the Total Variation (the sum of absolute values of the spatial gradients in the image plane) along the layers. See Figure 3. As can be seen, the Total Variation of the activations increases as we move deeper into the network. This means the activation map is less smooth and therefore the correlation between nearby activations is lower. As a result, we need to increase the ReLU count.

The previous set of experiments used patches to cluster activations. Next, we evaluate the performance of clustering. To this end, we clustered activations into k clusters based on the activation maps of all images in the training set. The clustering was done independently for each channel separately.

SVHN

Method	Patch size	#ReLU	Noise	Accuracy
Original	1×1	29×29	No	93.14%
Uniform	3×3	9×9	No	92.33%
Uniform	32×32	1	No	90.21%
Adaptive	32×32	256	No	91.63%
Noise	32×32	1	20%	91.22%
FC+Noise	32×32	1	20%	90.81%

Fashion-MNIST

Method	Patch size	#ReLU	Noise	Accuracy
Original	1×1	28×28	No	92.11%
Uniform	3×3	9×9	No	91.11%
Uniform	32×32	1	No	90.38%
Noise	32×32	1	20%	90.31%
FC+Noise	32×32	1	20%	90.20%

Table 6: SVHN and Fashion-MNIST ReLU sharing on image patches. The #ReLU column counts the number of ReLU operations per channel at the first layer of the network.

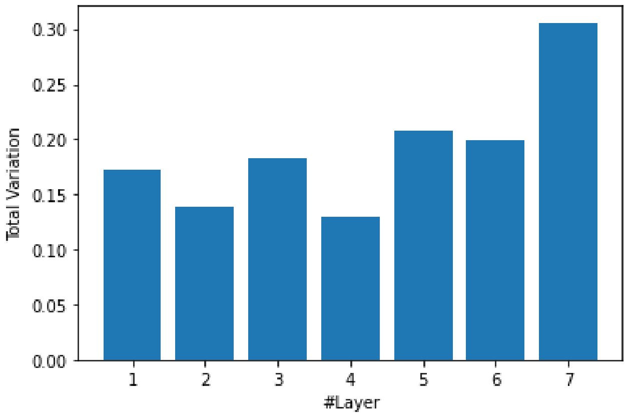

Figure 3: Total Variation (TV) across layers. We show the TV of the response images across the different layers of the network for the CIDAR-10 dataset. As can be seen, the TV increases with depth. We believe the two dips (at $x = 2, x = 4$) are because they are before an average pooling operation.

We use agglomerative clustering. First, we compute a binary activation vector per pixel, then we calculate the correlation matrix according to the hamming distance between the vectors. We initialize each activation as a set and in each iteration we take the two activations with the minimum hamming distance and if they are in different sets we unify their sets, and we stop when we have k clusters. The 64 clustering maps for the 64 channels in the second layer can be seen in Figure 4. We can see that many channels have highly correlated background pixels that are unified into a single cluster. The clustering maps reveal information about the *training* examples and *not* about the query image.

One can see the results of the clustering experiments in Table 7. In the first experiment we clustered every 8×8 window into 8 clusters so in total we have 128 clusters for each channel. In the second experiment we use image patches for the first layer where we take the middle pixel in each channel and for the second ReLU layer we use agglomerative clustering with 256 clusters for the whole channel so in total we have the same number of ReLUs in both experiments.

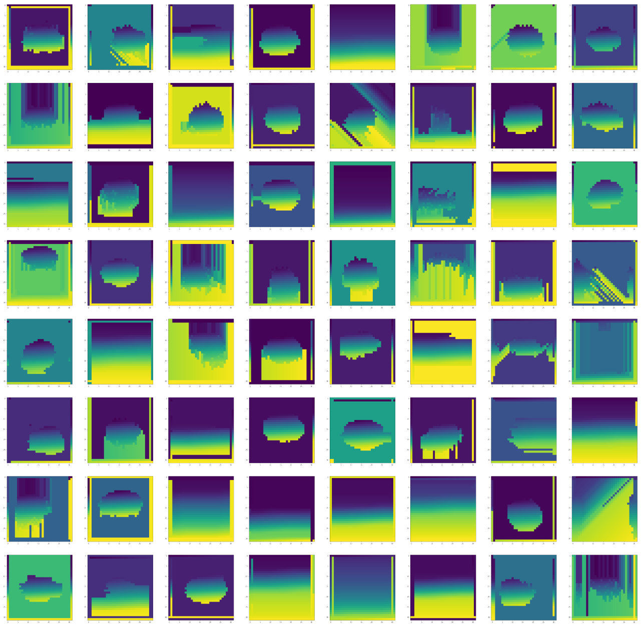

Figure 4: ReLU clustering. Clustering map of the 64 channels in the second layer of the CIFAR-10 network. Each channel is clustered into 256 clusters.

Method	Layer #1	Layer #2	#ReLU	Accuracy
# 1	$(8 \times 8, 8)$	$(8 \times 8, 8)$	256	81.96%
# 2	$(32 \times 32, 1)$	$(32 \times 32, 256)$	257	81.67%

Table 7: CIFAR-10 adaptive clustering results. The expression $(m \times m, k)$ means that we cluster every non-overlapping window of size $(m \times m)$ into k clusters. As can be seen, both methods use roughly the same number of ReLU operations. The first method outperforms the second one.

The results are slightly worse than the uniform 3×3 patch experiment reported earlier (See Table 5). The results of agglomerative clustering for SVHN can be seen in table 6.

We conclude that ReLU sharing across small image patches drops the ReLU count with minimal impact on accuracy. We further observe that agglomerative clustering does not outperform simply grouping activations into image patches. We hypothesize that this might be the case because nearby activations in the image plane are already highly correlated.

From ReLU Count to Communication Bandwidth Having established that dropping ReLU count has little impact on network accuracy, we investigate its impact on runtime, round complexity and communication bandwidth.

To do that, we evaluate 3 of the models introduced earlier that presented attractive ReLU count/accuracy trade-offs. This includes the original model, the Uniform with 32×32 patch size model (where we perform a single ReLU decision for each channel for the first 2 ReLU layers), and the FC + Noise model (where we perform a dot-product with a learned weight vector and add noise to the outcome). We can see the results for CIFAR-10 of classification of 1 test image in Table 8.

In the original model each of the 2 first ReLU layers takes 10 seconds and 46MB of communication bandwidth so just these 2 layers take over 55% of the total time and over 65% of the total communication bandwidth.

In the 32×32 Uniform model each of the 2 first ReLU layers takes 0.25 seconds and 2.66MB, compared to 10 seconds and 46MB of the original model. This leads to 60% savings in the total time and 61% savings in the total communication (the number of rounds stays the same).

The FC + Noise model takes slightly more communication from the 32×32 Uniform model (but still 59% less from the original model) since it has an extra fully connected layer and a multiplication with a sampled noise tensor but, as we saw in Table 5, this model has better accuracy so we can see here a trade off between accuracy and efficiency.

We observe similar behaviour for the SVHN data set. The first ReLU layer takes 3.7 seconds and 17.87MB of communication, so over 73% of the total time and over 80% of total communication. In the 32×32 Uniform model the first ReLU layer takes 0.21 seconds and 1.09MB, 70%-76% less than the original model costs.

For Fashion-MNIST, the first ReLU layer takes 3.49 seconds and 16.66MB of communication, so over 53% of the total time and over 47% of total communication. In the 32×32 Uniform model the first ReLU layer takes 0.2 seconds and 1.02MB, 45%-50% less than the original model costs. We can see the results for SVHN and Fashion-MNIST of classification of 1 test image in Table 8.

In all three models and across all three datasets we observe that drop in ReLU count translates directly into drop in communication bandwidth.

Comparison to other Approaches It is challenging to compare our method to other methods reported in the literature because we use different hardware and different cryptographic settings.

Specifically, we use the PySyft implementation that, in turn, relies on SecureNN (Wagh, Gupta, and Chandran 2019) protocols. These are 3-party protocols, as opposed to the 2-party protocols used by others, except for Chameleon (Riazi et al. 2018) that uses the third party only in the offline phase.

In our setting all 3 parties live on the same machine and do not communicate over the network, so all running times do not consider network latency. All other frameworks are evaluated in a LAN setting where each party has its own machine so their run time takes into consideration the network latency.

Previous works split their computation into an input independent offline phase and an input dependent online phase. The PySyft implementation of the secureNN protocols does not do this split and counts all cost as online cost – hence, the offline cost is 0. The online run time can be improved by

Dataset	Model	Accuracy	Runtime (s)	Rounds	Comm. (MB)
CIFAR-10	Original	83.59%	36	86	141.02
	Uniform 32×32	81.13%	14	86	54.18
	FC + Noise	82.01%	13.88	90	57.865
SVHN	Original	93.14%	5.03	26	22.1
	Uniform 32×32	90.21%	1.54	26	5.32
Fashion-MNIST	Original	92.11%	6.55	40	35.04
	Uniform 32×32	90.38%	3.26	40	19.04

Table 8: Classification performance. We compare the performance of three different models, in terms of run time, rounds of communication, and communication bandwidth. Results are for the classification of a single image. As can be seen, for CIFAR-10, both "Uniform 32×32" and "FC + Noise" cut run time and bandwidth by about 60% with little degradation in performance. For SVHN, the Uniform 32×32 model saves 70%-76% of the run time and communication costs and saves 45%-50% of the run time and communication costs in the Fashion-MNIST dataset.

Framework		**Accuracy**	**Runtime (s)**			**Comm. (MB)**		
			setup	online	total	setup	online	total
2PC	MiniONN	81.61%	472	72	544	3046	6226	9272
	GAZELLE	81.61%	15.5	4.25	19.8	906	372	1278
	FALCON	81.61%	10.5	3.31	13.8	906	372	1278
3PC	Chameleon	81.61%	22.97	29.7	52.67	1210	1440	2650
	Shafran *et al.*	83.53%	0	19.49	19.49	0	71.9	71.9
	OURS	82.01%	0	13.88	13.88	**0**	**57.86**	**57.86**

Table 9: Classification performance Comparison on CIFAR-10. We compare ourselves to MiniONN (Liu et al. 2017), GAZELLE (Juvekar, Vaikuntanathan, and Chandrakasan 2018), FALCON (Li et al. 2020), Chamelon (Riazi et al. 2018), and the crypto-oriented neural architecture design of (Shafran et al. 2019). The first three methods are 2-party MPC methods, while the last three methods take a 3-party MPC approach. In all cases our communication bandwidth is significantly lower than other methods. Our approach can be integrated into other algorithms and lead to similar savings.

performing the beaver triplet generation for multiplication in an offline phase (Beaver 1992).

We do observe that our work achieves a $\times 2.5$ improvement in online communication and a $\times 22$ improvement in total communication over the best results of GAZELLE (Juvekar, Vaikuntanathan, and Chandrakasan 2018) and FALCON (Li et al. 2020). In online run-time our work is roughly $\times 4$ slower than the best result of FALCON (Li et al. 2020) and slightly worse in total run-time (since our run-time doesn't take network latency into consideration).

Shafran *et al.* (Shafran et al. 2019) used different models in their paper so in order to compare our work with theirs we applied their method on the model in table 2. Specifically, we removed the first ReLU layer and replaced the second ReLU layer with 50% partial activation layer. See results in table 8. Their method does not hurt the accuracy compared to the original model, while our method requires 28% less time and 20% less communication bandwidth, at the cost of a roughly 1% drop in accuracy.

We believe that ReLU sharing can be implemented in other frameworks and improve their run time and communication cost. For example, we can see in the benchmarks table of FALCON (Li et al. 2020) for the ReLU operation that reducing the ReLU count by an order of magnitude leads to a roughly $\times 10$ drop in both time and communication costs, for both the offline and online stages. We also observe in the FC layer benchmark there that adding another FC layer will have a negligible effect on costs. This leads us to believe that implementing ReLU sharing within the FALCON, or many other algorithms, will decrease costs.

Conclusions

We propose a method to drastically reduce the ReLU count of a CNN with minimal impact on accuracy. The proposed method groups pixels (or activation units) together and uses a single ReLU operator for all activations in the group. We discuss how to group activations and how to apply the ReLU operation.

Experiments suggest that we can reduce the ReLU count by up to three orders of magnitude which, in turn, cuts the communication bandwidth by more than 50%. Our approach can be used with other Privacy Preserving Machine Learning algorithms and bring us closer to practical algorithms that can be used in the wild.

Acknowledgments This research was supported in part by a grant made by TATA Consultancy Services.

References

Beaver, D. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In Feigenbaum, J., ed., *Advances in Cryptology — CRYPTO '91*, 420–432. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-46766-3.

Dowlin, N.; Gilad-Bachrach, R.; Laine, K.; Lauter, K.; Naehrig, M.; and Wernsing, J. 2016. CryptoNets: Applying

Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, 201–210. JMLR.org.

Goldreich, O.; Micali, S.; and Wigderson, A. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In Aho, A. V., ed., *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, 218–229. ACM. doi: 10.1145/28395.28420.

Juvekar, C.; Vaikuntanathan, V.; and Chandrakasan, A. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *Proceedings of the 27th USENIX Conference on Security Symposium*, SEC'18, 1651–1668. USA: USENIX Association. ISBN 9781931971461.

Li, S.; Xue, K.; Zhu, B.; Ding, C.; Gao, X.; Wei, D.; and Wan, T. 2020. FALCON: A Fourier Transform Based Approach for Fast and Secure Convolutional Neural Network Predictions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

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*, CCS '17, 619–631. New York, NY, USA: Association for Computing Machinery. ISBN 9781450349468. doi:10.1145/3133956.3134056. URL https://doi.org/10.1145/3133956.3134056.

Mohassel, P.; and Zhang, Y. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, 19–38.

OpenMined. 2018. PySyft: A Python library for secure and private Deep Learning. In *https://github.com/OpenMined/PySyft*.

Riazi, M. S.; Weinert, C.; Tkachenko, O.; Songhori, E. M.; Schneider, T.; and Koushanfar, F. 2018. Chameleon: A Hybrid Secure Computation Framework for Machine Learning Applications. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, ASIACCS '18, 707–721. New York, NY, USA: Association for Computing Machinery. ISBN 9781450355766. doi: 10.1145/3196494.3196522. URL https://doi.org/10.1145/3196494.3196522.

Rouhani, B. D.; Riazi, M. S.; and Koushanfar, F. 2018. Deepsecure: Scalable Provably-Secure Deep Learning. In *Proceedings of the 55th Annual Design Automation Conference*, DAC '18. New York, NY, USA: Association for Computing Machinery. ISBN 9781450357005. doi:10.1145/3195970.3196023.

Shafran, A.; Segev, G.; Peleg, S.; and Hoshen, Y. 2019. Crypto-Oriented Neural Architecture Design. *CoRR* abs/1911.12322. URL http://arxiv.org/abs/1911.12322.

Wagh, S.; Gupta, D.; and Chandran, N. 2019. SecureNN: 3-Party Secure Computation for Neural Network Training. In *Proceedings on Privacy Enhancing Technologies (PETS)*.

Yao, A. C. 1982. Protocols for Secure Computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, 160–164. USA: IEEE Computer Society.