Scalable and Efficient Training of Large Convolutional Neural Networks with Differential Privacy

Anonymous Author(s)
Affiliation
Address
email

Abstract

Large convolutional neural networks (CNN) can be difficult to train in the differentially private (DP) regime, since the optimization algorithms require a computationally expensive operation, known as the per-sample gradient clipping. We propose an efficient and scalable implementation of this clipping on convolutional layers, termed as the mixed ghost clipping, that significantly eases the private training in terms of both time and space complexities, without affecting the accuracy. The improvement in efficiency is rigorously studied through the first complexity analysis for the mixed ghost clipping and existing DP training algorithms.

Extensive experiments on vision classification tasks, with large ResNet, VGG, and Vision Transformers, demonstrate that DP training with mixed ghost clipping adds $1 \sim 10\%$ memory overhead and $< 2\times$ slowdown to the standard non-private training. Specifically, when training VGG19 on CIFAR10, the mixed ghost clipping is $3\times$ faster than state-of-the-art Opacus library with $18\times$ larger maximum batch size. To emphasize the significance of efficient DP training on convolutional layers, we achieve 96.7% accuracy on CIFAR10 and 83.0% on CIFAR100 at $\epsilon = 1$ using BEiT, while the previous best results are 94.8% and 67.4%, respectively. We open-source a privacy engine that implements DP training of CNN with a few lines of code.

1 Introduction

Deep convolutional neural networks (CNN) [16, 30] are the backbone in vision-related tasks, including image classification [28], object detection [37], image generation [19], video recognition [39], and audio classification [22]. A closer look at the dominating success of deep CNNs reveals its basis on two factors.

The first factor is the strong capacity of the convolutional neural networks, usually characterized by the enormous model size. Recent state-of-the-art progresses usually result from very large models, with millions to billions of trainable parameters. For example, ImageNet [10] accuracy grows when larger VGGs [40] (increasing 11 to 19 layers$\Longrightarrow$69% to 74% accuracy) or ResNets [20] (increasing 18 to 152 layers$\Longrightarrow$70% to 78% accuracy) are used [42]. Consequently, the eager to use larger models for better accuracy naturally draws people's attention to the scalability and efficiency of training.

The second factor is the availability of big data, which oftentimes contain private and sensitive information. The usage of such data demands rigorous protection against potential privacy attacks. In fact, one standard approach to guarantee the protection is by differentially private (DP) [13, 14] training of the models. Since [1], CNNs have achieved promising results under strong DP guarantee: CIFAR10 achieves 92.4% accuracy in [45] and ImageNet achieves 81.1% accuracy in [9].

Unifying the two driving factors of CNNs leads to the DP training of large CNNs. However, the following challenges are hindering our application of large and private CNNs in practice.

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

Challenge 1: Time and space efficiency in DP training. DP training can be extremely inefficient in memory and speed. For example, a straightforward implementation in Tensorflow Privacy library shows that DP training can be $1000\times$ slower than the non-DP training, even on a small RNN [5]; other standard DP libraries, Opacus [49] and JAX [29, 41], which trade off memory for speed, could not fit a single datapoint into GPU on GPT2-large [32]; addtionally, $3 \sim 9\times$ slowdown of DP training has been reported in [29, 9, 41] using JAX.

The computational bottleneck comes from the per-sample gradient clipping at each iteration (see (2.1)), a necessary step in DP deep learning. I.e., denoting the loss as $\sum_i \mathcal{L}_i$, we need to clip the per-sample gradient $\{\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}\}_i$ individually. This computational issue is even more severe when we apply a large batch size, which is necessary to achieve high accuracy of DP neural networks. In [32, 29, 33], it is shown that the optimal batch size for DP training is significantly larger than for regular training. For instance, DP ResNet18 achieves best performance on ImageNet when batch size is 64*1024 [29]; and DP ResNet152 and ViT-Large use a batch size 2^{20} in [33]. As a result, an efficient implementation of per-sample gradient clipping is much-needed to fully leverage the benefit of large batch training.

Challenge 2: Do large DP vision models necessarily harm accuracy? An upsetting observation in DP vision models is that, over certain relatively small model size, larger DP CNNs seem to underperform smaller ones. This is observed in models that are either pre-trained or trained from scratch [25, 1]. As an example of the pre-trained cases, previously state-of-the-art CIFAR10 is obtained from a small DP linear model [45], and the fine-tuned DP ResNet50 underperforms DP ResNet18 on ImageNet [29]. On the contrary, the empirical evidence in DP language models shows that larger models can consistently achieve better accuracy [32]. Interestingly, we empirically demonstrate that the trend in [32] can possibly hold in vision models as well.

1.1 Contributions

In this work, we propose new algorithms to efficiently train large-scale CNNs with DP optimizers. To be specific, our contributions are as follows.

1. We propose a novel implementation, termed as the ***mixed ghost clipping***, of the per-sample gradient clipping for 1D$\sim$3D convolutional layers. The mixed ghost clipping is the first method that can ***implement per-sample gradient clipping without per-sample gradients*** of the convolutional layers. It works with any DP optimizer and any clipping function almost as memory efficiently as in standard training, thus significantly outperforming existing implementation like Opacus [49].
2. In some tasks, mixed ghost clipping also claims supremacy on speed using a fixed batch size. The speed can be further boosted (say $1.7\times$ faster than the fastest alternative DP algorithms and only $2\times$ slower than the non-private training) when the memory saved by our method is used to fit the largest possible batch size.
3. We provide the first complexity analysis of mixed ghost clipping in comparison to other training algorithms. This analysis clearly indicates the necessity of our layerwise decision principle, without which the existing methods suffer from high memory burden.
4. Leveraging our algorithms, we can efficiently train large DP models, such as VGG, ResNet, Wide-ResNet, DenseNet, and Vision Transformer (ViT). Using DP ViTs at ImageNet scale, we are the first to train convolutional ViTs under DP and achieve dominating SOTA on CIFAR10/100 datasets, thus bringing new insights that larger vision models can consistently achieve better accuracy under DP.

1.2 Previous arts

The straightforward yet highly inefficient way of per-sample gradient clipping is to use batch size 1 and compute gradients with respect to each individual loss. Recently, more advanced methods have significantly boosted the efficiency by avoiding such a naive approach. The most widely applied method is implemented in the Opacus library [49], which is fast but memory costly as per-sample gradients $\boldsymbol{g}_i = \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}$ are instantiated to compute the weighted gradient $\sum_i C_i \cdot \boldsymbol{g}_i$ in (2.1). A more efficient method, FastGradClip [31], is to use a second back-propagation with weighted loss $\sum_i C_i \cdot \mathcal{L}_i$ to indirectly derive the weighted gradient.

In all above-mentioned methods and [38], the per-sample gradients are instantiated, whereas this can be much inefficient and not necessary according to the 'ghost clipping' technique, as to be detailed in Section 3. In other words, ghost clipping proves that the claim 'DP optimizers require access to the per-sample gradients' is wrong. Note that ghost clipping is firstly proposed by [18] for linear layers, and then extended by [32] to sequential data and embedding layers for language models. However, the ghost clipping has not been extended to convolutional layers, due to the complication of the convolution operation and the high dimension of data (text data is mostly 2D, yet image data are 3D and videos are 4D). In fact, we will show that even ghost clipping alone is not satisfactory for CNNs, and therefore propose the mixed ghost clipping, that narrows the efficiency gap between DP training and the regular training.

2 Preliminaries

2.1 Differential privacy

Differential privacy (DP) has become the standard approach to provide privacy guarantee for modern machine learning models. The privacy level is characterized through a pair of privacy quantities (ϵ, δ), where smaller (ϵ, δ) means stronger protection.

Definition 2.1 ([14]). A randomized algorithm M is (ε, δ)-DP if for any neighboring datasets S, S' that differ by one arbitrary sample, and for any event E, it holds that

$$\mathbb{P}[M(S) \in E] \leqslant \mathrm{e}^{\varepsilon} \mathbb{P}\left[M\left(S'\right) \in E\right] + \delta.$$

In deep learning where the number of parameters are large, the Gaussian mechanism [14, Theorem A.1] is generally applied to achieve DP at each training iteration, i.e. we use regular optimizers on the following privatized gradient:

$$\widetilde{\boldsymbol{g}} = \sum_i C(\|\boldsymbol{g}_i\|; R) \cdot \boldsymbol{g}_i + \sigma R \cdot \mathcal{N}(0, \mathbf{I}) = \sum_i C_i \boldsymbol{g}_i + \sigma R \cdot \mathcal{N}(0, \mathbf{I}) \tag{2.1}$$

where C is any function whose output is upper bounded by $R/\|\boldsymbol{g}_i\|$ and R is known as the clipping norm. To name a few examples of C, we have the Abadi's clipping $\min(R/\|\boldsymbol{g}_i\|, 1)$ in [1] and the global clipping $\mathbb{I}(\|\boldsymbol{g}_i\| < Z) \cdot R/Z$ for any constant Z in [6]. Here σ is the noise multiplier that affects the privacy loss (ϵ, δ), but R only affects the convergence, not the privacy.

In words, DP training switches from updating with $\sum_i \boldsymbol{g}_i$ to updating with the private gradient $\widetilde{\boldsymbol{g}}$: SGD with private gradient is known as DP-SGD; Adam with private gradient is known as DP-Adam.

Algorithmically speaking, the Gaussian mechanism can be decomposed into two parts: the per-sample gradient clipping and the Gaussian noise addition. From the viewpoint of computational complexity, the per-sample gradient clipping is the bottleneck, while the noise addition costs negligible overhead.

In this work, our focus is the implementation of per-sample gradient clipping (2.1). We emphasize that our implementation is only on the algorithmic level, not affecting the mathematics and thus not the performance of DP optimizers. That is, our mixed ghost clipping provides exactly the same accuracy results as Opacus, FastGradClip, etc.

2.2 Per-sample gradient for free during standard back-propagation

In DP training, the per-sample gradient is a key quantity which can be derived for free from the standard back-propagation. We briefly introduce the back-propagation on linear layers, following the analysis from [18, 32], so as to prepare our new clipping implementation for convolutional layers. Note that the convolutional layers can be viewed as equivalent to the linear layers in Section 2.3.

Let the input of a hidden layer be $\mathbf{a} \in \mathbb{R}^{B \times \cdots \times d}$ (a.k.a. post-activation). Here $\mathbf{a}$ can be in high dimension: for sequential data such as text, $\mathbf{a} \in \mathbb{R}^{B \times T \times d}$ where T is the sequence length; for image data, $\mathbf{a} \in \mathbb{R}^{B \times H \times W \times d}$ where (H, W) is the dimension of image and d is number of channels; for 3D objects or video data, $\mathbf{a} \in \mathbb{R}^{B \times H \times W \times D \times d}$ where D is the depth or time length, respectively.

Denote the weight of a linear layer as $\mathbf{W} \in \mathbb{R}^{d \times p}$, its bias as $\mathbf{b} \in \mathbb{R}^p$ and its output (a.k.a. pre-activation) as $\mathbf{s} \in \mathbb{R}^{B \times \cdots \times p}$, where B is the batch size and p is the output dimension.

In the l-th layer of a neural network with L layers in total, we denote its weight, bias, input and output as $\mathbf{W}_{(l)}, \mathbf{b}_{(l)}, \mathbf{a}_{(l)}, \mathbf{s}_{(l)}$ respectively, and the activation function as ϕ. Consider

$$\mathbf{a}_{(l+1),i} = \phi(\mathbf{s}_{(l),i}) = \phi(\mathbf{a}_{(l),i}\mathbf{W}_{(l)} + \mathbf{b}_{(l)}). \tag{2.2}$$

Clearly the i-th sample's *hidden feature* $\mathbf{a}_{(l),i}$ at layer l is freely extractable during the forward pass.

Let $\mathcal{L} = \sum_{i=1}^{n} \mathcal{L}_i$ be the total loss and $\mathcal{L}_i$ be the per-sample loss with respect to the i-th sample. During a standard back-propagation, the following *partial product* is maintained,

$$\frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{a}_{(L),i}} \circ \frac{\partial \mathbf{a}_{(L),i}}{\partial \mathbf{s}_{(L-1),i}} \cdot \frac{\partial \mathbf{s}_{(L-1),i}}{\partial \mathbf{a}_{(L-1),i}} \circ \cdots \frac{\partial \mathbf{a}_{(l+1),i}}{\partial \mathbf{s}_{(l),i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l+1),i}} \mathbf{W}_{(l+1)} \circ \phi'(\mathbf{s}_{(l),i}) \tag{2.3}$$

so as to compute the standard gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{(l)}} = \sum_i \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_{(l)}}$ in (2.4). Here $\circ$ is the Hadamard product and $\cdot$ is the matrix product. Therefore, $\frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}}$ is also available for free from (2.3) and extractable by Pytorch hooks, which allows us compute the per-sample gradient by

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}_{(l)}} = \frac{\partial \mathcal{L}_i}{\partial \mathbf{s}_{(l),i}}^\top \frac{\partial \mathbf{s}_i}{\partial \mathbf{W}_{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}}^\top \mathbf{a}_{(l),i}, \quad \frac{\partial \mathcal{L}_i}{\partial \mathbf{b}_{(l)}} = \frac{\partial \mathcal{L}_i}{\partial \mathbf{s}_{(l),i}}^\top \frac{\partial \mathbf{s}_{(l),i}}{\partial \mathbf{b}_{(l)}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}}^\top \mathbf{1}. \tag{2.4}$$

2.3 Equivalence between convolutional and linear layer

In a convolutional layer[1], the forward pass is

$$\mathbf{a}_{(l+1),i} = \phi(\mathbf{s}_{(l),i}) = \phi(F(U(\mathbf{a}_{(l),i})\mathbf{W}_{(l)} + \mathbf{b}_{(l)})) \tag{2.5}$$

in which F is the folding operation and U is the unfolding operation. To be clear, we consider a 2D convolution that $\mathbf{a}_{(l,i)} \in \mathbb{R}^{H_{\text{in}} \times W_{\text{in}} \times d_{(l)}}$ is the input of hidden feature, $(H_{\text{in}}, W_{\text{in}})$ is the input dimension, $d_{(l)}$ is the number of input channels. Then U unfolds the hidden feature from dimension $(H_{\text{in}}, W_{\text{in}}, d_{(l)})$ to $(H_{\text{out}}W_{\text{out}}, d_{(l)}k_H k_W)$, where k_H, k_W are the kernel sizes and $(H_{\text{out}}, W_{\text{out}})$ is the output dimension. After the matrix multiplication with $\mathbf{W}_{(l)} \in \mathbb{R}^{d_{(l)}k_H k_W \times p_{(l)}}$, the intermediate output $\mathbf{s}_{(l),i}$ is folded by F from dimension $(H_{\text{out}}W_{\text{out}}, p_{(l)})$ to $(H_{\text{out}}, W_{\text{out}}, p_{(l)})$.

To present concisely, we ignore the layer index l and write the per-sample gradient of weight for the convolutional layer, in analogy to the linear layer in (2.4),

$$\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} = \frac{\partial \mathcal{L}}{\partial F^{-1}(\mathbf{s}_i)}^\top U(\mathbf{a}_i) = F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right)^\top U(\mathbf{a}_i). \tag{2.6}$$

Here F^{-1} is the inverse operation of F and simply flattens all dimensions except the last one: from $(H_{\text{out}}, W_{\text{out}}, p_{(l)})$ to $(H_{\text{out}}W_{\text{out}}, p_{(l)})$. From (2.6), we derive the per-sample gradient norm for the convolutional layers from the same formula as in [32, Appendix F],

$$\left\| \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}} \right\|_{\text{Fro}}^2 = \text{vec}(U(\mathbf{a}_i)U(\mathbf{a}_i)^\top)\text{vec}\left(F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right) F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right)^\top\right). \tag{2.7}$$

3 Ghost clipping for Convolutional Layers

Leveraging our derivation in (2.7), we propose the ghost clipping to compute the clipped gradient without ever generating the per-sample gradient $\frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}$. The entire procedure is comprised of the ghost norm computation and the second back-propagation, as demonstrated in Figure 1.

3.1 Ghost norm: computing gradient norm without the gradient

The per-sample gradient norm is required to compute the per-sample C_i in (2.1). While it is natural to instantiate the per-sample gradients and then compute their norms [38, 31, 49, 9, 33], this is not always optimal nor necessary. Instead, we can leverage (2.7), the ghost norm, to compute the per-sample gradient norm and avoid the possibly expensive per-sample gradient. Put differently, when $T = H_{\text{out}}W_{\text{out}}$ is small, the multiplication $U(\mathbf{a}_i)U(\mathbf{a}_i)^\top$ plus $F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right) F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right)^\top$ is cheap, but the multiplication $F^{-1}\left(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_i}\right)^\top U(\mathbf{a}_i)$ is expensive. We demonstrate the ghost clipping's supremacy over complexity empirically in Table 4 and theoretically in Table 2.

[1] See a detailed explanation in Appendix C for the U, F operation and the dimension formulae in convolution.

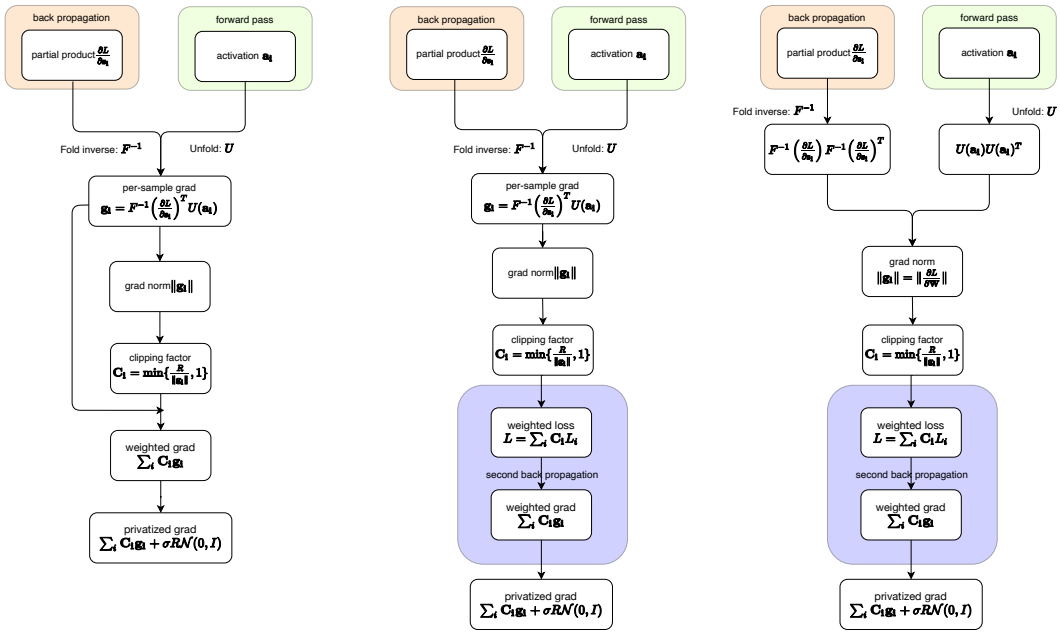

Figure 1: Per-sample gradient clipping for convolutional layers. **Left: Opacus** = Back-propagation + Gradient instantiation + Weighted gradient. **Middle: FastGradClip** = Back-propagation + Gradient instantiation + Second back-propagation. **Right: Ghost clipping** = Back-propagation + Ghost norm + Second back-propagation. See Section 4.1 for their complexity analysis.

3.2 Second back-propagation: weighted loss leads to weight gradient

We conduct a second back-propagation with the weighted loss $\sum_i C_i \mathcal{L}_i$ to derive the weighted gradient $\sum_i C_i \boldsymbol{g}_i$ in (2.1), which costs extra time. In contrast, Opacus [49] and JAX [9, 41] generate and store the per-sample gradient $\boldsymbol{g}_i$ for all $i \in [B]$. Thus the weighted gradient is directly computable from $\boldsymbol{g}_i$ as the memory is traded off for faster computation. However, in some cases like Table 7 on ImageNet and Table 9 on CIFAR100, we can use larger batch size to compensate the slowdown of the second back-propagation.

4 Mixed Ghost Clipping: To be a ghost or not, that is the question

Algorithm 1 Mixed Ghost Clipping (single iteration)

Parameters: number of layers L, gradient clipping norm R.

for $l = 1, 2, \ldots, L$ **do**
  if $2T_{(l)}^2 < p_{(l)} d_{(l)}$ **then**
    $\mathbf{W}_{(l)}$.ghost_norm = True ▷ Forward pass
  Compute $\mathbf{a}_{(l+1),i} = \phi(F(U(\mathbf{a}_{(l),i})\mathbf{W}_{(l)} + \mathbf{b}_{(l)}))$

Compute per-sample losses $\mathcal{L}_i$.

for $l = L, L-1, \ldots, 1$ **do**
  Compute $\frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}} = \frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l+1),i}} \mathbf{W}_{(l+1)} \circ \phi'(\mathbf{s}_{(l),i})$ ▷ Mixed ghost norm in first back-propagation
  if $\mathbf{W}_{(l)}$.ghost_norm = True **then**
    $\|\frac{\partial \mathcal{L}_i}{\partial W_{(l)}}\|_{\text{Fro}}^2 = \text{vec}(U(\mathbf{a}_{(l),i})U(\mathbf{a}_{(l),i})^\top)\text{vec}\left(\frac{\partial \mathcal{L}}{\partial F^{-1}(\mathbf{s}_{(l),i})} \frac{\partial \mathcal{L}}{\partial F^{-1}(\mathbf{s}_{(l),i})}^\top\right)$
  else
    $\frac{\partial \mathcal{L}_i}{\partial W_{(l)}} = F^{-1}(\frac{\partial \mathcal{L}}{\partial \mathbf{s}_{(l),i}})U(\mathbf{a}_{(l),i}) \longrightarrow \|\frac{\partial \mathcal{L}_i}{\partial W_{(l)}}\|_{\text{Fro}}^2$

Compute per-sample gradient norm $\|\frac{\partial \mathcal{L}_i}{\partial W}\|_{\text{Fro}}^2 = \sum_l \|\frac{\partial \mathcal{L}_i}{\partial W_{(l)}}\|_{\text{Fro}}^2$

Compute weighted loss $\mathcal{L}_{\text{weighted}} = \sum_i C(\|\frac{\partial \mathcal{L}_i}{\partial W}\|_{\text{Fro}}; R) \cdot \mathcal{L}_i$

Second back-propagation with $\mathcal{L}_{\text{weighted}}$ to generate $\sum_i C_i \frac{\partial \mathcal{L}_i}{\partial \mathbf{W}}$

While the ghost norm offers the direct computation gradient norm at the cost of an indirect computation of the weighted gradient, we will show that ghost clipping alone may not be sufficient for efficient DP training, as we demonstrate in Table 4 and Figure 3. In Table 2, we give the first fine-grained analysis of the space and time complexity for DP training algorithms. Our analysis gives the precise condition when the per-sample gradient instantiation (adopted in Opacus [49]) is more or less efficient than our ghost norm method. To take the advantage of both methods, we propose the mixed ghost clipping method in Algorithm 1, which applies the ghost clipping or non-ghost clipping by a layerwise decision.

We highlight that the key reason supporting the success of mixed ghost clipping method is its layerwise adaptivity to the dimension parameters, $(p_{(l)}, d_{(l)}, T_{(l)}, k_H, k_W)$, which vary largely across different layers (see Figure 2). The variance results from the fact that images are non-sequential data, and that the convolution and pooling can change the size ($T = H_{\text{out}} W_{\text{out}}$) of hidden features drastically.

In the next two sections, we will analyze rigorously the time and memory complexities of the regular training and the DP training, using ghost or non-ghost clippings.

Remark 4.1. In Algorithm 1, we present the mixed ghost clipping that prioritizes the space complexity by (4.1). We also derive and implement a speed-priority version by comparing the time complexity of ghost norm and gradient instantiation in Table 1. However, the efficiency difference is empirically insignificant.

4.1 Complexity analysis

We now break each clipping method into operation modules and analyze their complexities. A similar but coarse analysis from [32] only claims, on sequential layers, $O(BT^2)$ space complexity with ghost clipping and $O(Bpd)$ without ghost clipping. The time complexity and/or convolutional layers are not analyzed until this work.

Complexity	Back-propagation	Ghost norm	Grad instantiation	Weighted grad
Time	$2BTD(2p+1)$	$2BT^2(D+p+1)-B$	$[2B(T+2)-1]pD$	$2BpD$
Space	$BTp+2BTD+pD$	$B(2T^2+1)$	$B(pD+1)$	0

Table 1: Complexities of operation modules in per-sample gradient clipping methods, contributed by a single 2D convolutional layer.

Here B is the batch size, $D = dk_H k_W$ where d is the number of input channels, k is the kernel sizes, p is the number of output channels, and $T = H_{\text{out}} W_{\text{out}}$. We leave the detailed complexity computation in Appendix D. Leveraging Table 1, we give the complexities of different clipping algorithms in Table 2.

Complexity	Time	Space
Opacus [49]	$6BTpD$	$B(pd+Tp+2TD)$*
FastGradClip [31]	$10BTpD$	$B(pD+Tp+2TD)$
Ghost clipping (ours)	$8BTpD+2BT^2(p+D)$	$B(2T^2+Tp+2TD)$
Mixed ghost clipping (ours)	SEE CAPTION	$B(\min(2T^2,pD)+Tp+2TD)$

Table 2: Complexity of different implementations of per-sample gradient clipping by a single 2D convolutional layer. Only highest order terms are listed. * indicates that Opacus stores the per-sample gradients of all layers, thus a per-layer space complexity does not accurately characterize its memory burden, since other methods only store the intermediate variables one layer at a time. The mixed ghost clipping's time complexity is between FastGradClip and ghost clipping, depending on which of $(2T^2, pD)$ is smaller.

4.2 Layerwise decision in mixed clipping

From the space complexity in Table 2, we derive the layerwise decision that selects the more memory efficient of FastGradClip (gradient instantiation) and ghost clipping (ghost norm):

$$\text{Choose ghost norm over per-sample gradient instantiation if } 2T^2 < pD = pdk_H k_W. \tag{4.1}$$

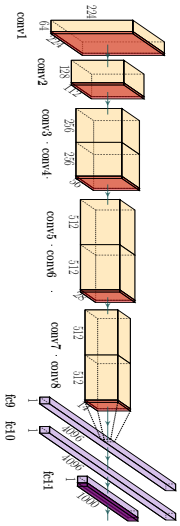

Figure 2: VGG-11 architecture on ImageNet (224×224).

Space Complexity	Ghost norm $2T_{(l)}^2 = 2H_{\text{out}}^2 W_{\text{out}}^2$	Non-ghost norm $p_{(l)} d_{(l)} k_H k_W$
conv1	5.0×10^9	$\mathbf{1.7 \times 10^3}$
conv2	3.0×10^8	$\mathbf{7.3 \times 10^4}$
conv3	2.0×10^7	$\mathbf{2.9 \times 10^5}$
conv4	2.0×10^7	$\mathbf{5.8 \times 10^5}$
conv5	1.2×10^6	$\mathbf{1.1 \times 10^6}$
conv6	$\mathbf{1.2 \times 10^6}$	2.3×10^6
conv7	$\mathbf{7.6 \times 10^4}$	2.3×10^6
conv8	$\mathbf{7.6 \times 10^4}$	2.3×10^6
fc9	$\mathbf{2}$	1.0×10^8
fc10	$\mathbf{2}$	1.6×10^7
fc11	$\mathbf{2}$	4.1×10^6

Table 3: Layerwise decision of mixed ghost clipping on VGG-11. Green background indicates being selected.

Therefore, our mixed ghost clipping is a mixup of FastGradClip and the ghost clipping (c.f. Figure 1). We note that the decision by the mixed ghost clipping (4.1) depends on different dimensions: the ghost clipping depends on the size of hidden features (height H and width W) which in turn depends on kernel size, stride, dilation and padding (see Appendix C for introduction of convolution), while only the non-ghost clipping depends on the number of channels. In ResNet and VGG, the hidden feature size decreases as layer depth increases, due to the shrinkage from the convolution and pooling operation; on the opposite, the number of channels increases in deeper layers.

Remark 4.2 (Ghost clipping favors bottom layers). As a consequence of decreasing hidden feature size and increasing number of channels, there exists a depth threshold beyond which the ghost clipping is preferred in bottom layers, where the save in complexity is substantial. In Figure 2 and Table 3, as the layer of VGG 11 goes deeper, the hidden feature size shrinks from $224 \to 112 \to \cdots \to 14$ and the number of channels increases from $3 \to 64 \to \cdots \to 512$.

5 Performance

We compare our ghost clipping and mixed ghost clipping methods to state-of-the-art clipping algorithms, namely Opacus [49] and FastGradClip [31], which are implemented in Pytorch. We are aware of but will not compare to implementations of these two algorithms in JAX [4], e.g. [29, 41, 9], so as to only focus on the algorithms rather than the operation framework. All experiments run on one Tesla V100 GPU (16GB RAM).

We highlight that switching from the regular training to DP training only needs a few lines of code using our privacy engine (see Appendix F). For CNNs, we use models from `https://github.com/kuangliu/pytorch-cifar` on CIFAR10 [27] and models from Torchvision [42] on ImageNet [10]. For ViTs, regardless of datasets, we use models from PyTorch Image Models [46].

5.1 Time and memory efficiency (fixed batch size)

We first measure the time and space complexities when the physical batch size is fixed. Here we define the physical batch size (or the virtual batch size) as the number of samples actually fed into the memory, which is different from the logical batch size. For example, if we train with batch size 1000 but can only feed 40 samples to GPU at one time, we back-propagate 25 times before updating the weights for 1 time. This technique is known as the gradient accumulation and is widely applied in large batch training, which particularly benefits the accuracy of DP training [32, 29, 9, 33].

From Table 4 and the extended Table 6, we see a clear advantage of mixed ghost clipping: our clipping only incurs $\leq 1\%$ memory overhead than the regular training, and is the fastest DP training algorithm. In contrast on ResNet18, Opacus uses $5\times$ memory, and FastGradClip uses $2\times$ memory. Even the ghost clipping uses $1.2\times$ memory of regular training, while being slower than both Opacus and FastGradClip. Similar phenomenon is observed on ImageNet in Table 7.

Dataset	Model & # Params	Package	Time (sec) / Epoch	Active Memory (GB)
CIFAR10	CNN [45, 35] 0.551M	Opacus	12	1.37
		FastGradClip	11	0.94
		Ghost (ours)	11	2.47
		Mixed (ours)	7	0.79
		Non-DP	5	0.66
	ResNet 18 / 34 / 50 11M / 21M / 23.5M	Opacus	OOM / OOM / OOM	OOM / OOM / OOM
		FastGradClip	45 / 80 / OOM	6.32 / 7.65 / OOM
		Ghost (ours)	59 / 98 / 158	4.00 / 4.90 / 9.62
		Mixed (ours)	37 / 66 / 119	3.31 / 4.13 / 9.62
		Non-DP	14 / 24 / 49	3.30 / 4.12 / 9.56
	VGG 11 / 13 / 16 9M / 9.4M / 14.7M	Opacus	OOM / OOM / OOM	OOM / OOM / OOM
		FastGradClip	18 / 25 / 33	5.17 / 5.45 / 5.61
		Ghost (ours)	14 / 25 / 29	2.58 / 3.30 / 3.41
		Mixed (ours)	13 / 18 / 23	2.58 / 2.76 / 2.84
		Non-DP	5 / 6 / 8	2.54 / 2.73 / 2.81

Table 4: Time and memory of selected models on CIFAR10, with physical batch size 256. Additional models are in Table 6. Out of memory (OOM) means the total memory exceeds 16GB.

5.2 Maximum batch size and throughput

Importantly, the speed efficiency in Table 4 can be further boosted, if we use up the saved memory to increase the batch size. To stress test the maximum physical batch size and the throughput of each clipping method, we train ResNet [20], VGG [40], MobileNet[23], ResNeXt [47],AlexNet[26],Wide-ResNet[51], DenseNet[24] and ViTs on CIFAR10 and ImageNet, as summarized partially in Figure 3 and in Table 7, respectively. For example, on VGG19 and CIFAR10, the mixed ghost clipping has a maximum batch size $18\times$ bigger (thus $3\times$ faster) than Opacus, $3\times$ bigger (thus $1.7\times$ faster) than FastGradClip, and $2\times$ bigger (thus $1.3\times$ faster) than the ghost clipping. Similarly, on Wide-ResNet50 and ImageNet, the mixed ghost clipping has a maximum batch size $5\times$ bigger than Opacus, and $< 0.3\%$ more memory costly than the non-private training.

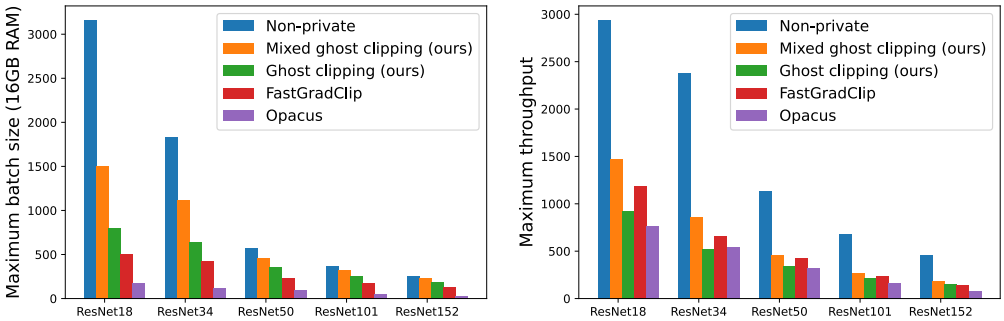

Figure 3: Memory (left) and speed (right) comparison of DP clipping algorithms on CIFAR10.

5.3 Vision transformers with convolution on ImageNet scale

In addition to training large-scale CNNs such as ResNet152, we apply our mixed ghost clipping to train ViTs, which substantially outperform existing SOTA on CIFAR10 and CIFAR100. Notice that the ViTs are pretrained on ImageNet scale, by which we resize CIFAR images (from 32×32 pixels to 224×224 pixels).

It is worth mentioning that ViT [12] is originally proposed as a substitute of CNN. Hence it and many variants do not contain convolutional layers. Here we specifically consider the convolutional ViTs, including ScalableViT[48], XCiT[2], Visformer[8], CrossVit[7], NesT[53], CaiT[44], DeiT[43], BEiT[3], PiT[21], and ConViT[15].

Performance of these ViTs on CIFAR10 and CIFAR100 are listed in Appendix E for a single-epoch DP training and several ViTs already beat previous SOTA, even though we do not apply additional techniques as in [9, 33] (e.g. learning rate schedule or random data augmentation).

	ε	Test Accuracy (%)	
		CIFAR-10	CIFAR-100
Yu et al. [50] (ImageNet1k)	1	94.3	–
	2	94.8	–
Tramer et al. [45] (ImageNet1k)	2	92.7	–
De et al. [9] (ImageNet1k)	1	94.8	67.4
	2	95.4	74.7
	4	96.1	79.2
	8	96.6	81.8
Our CrossViT base (104M params) (ImageNet1k)	1	95.5	71.9
	2	96.1	74.3
	4	96.2	76.7
	8	96.5	77.8
Our BEiT large (303M params) (ImageNet21k)	1	96.7	83.0
	2	97.1	86.2
	4	97.2	87.7
	8	97.4	88.4

Table 5: CIFAR-10 and CIFAR-100 test accuracies when fine-tuning with DP-SGD. We train CrossViT base for 5 epochs, lr 0.002, optim. We train BEiT large for 3 epochs lr 0.001. Here batch size is 1000 and clipping norm is 0.1. '( )' indicates the pretrained datasets.

By training multiple epochs with best performing ViTs in Table 8 and Table 9, we achieve new SOTA under DP in Table 5, with substantial improvement especially for strong privacy guarantee (e.g. $\epsilon < 2$). Our DP training is at most $2\times$ slower and 10% more memory expensive than the non-private training, even on BEiT large, thus significantly improving the $9\times$ slowdown reported in [9].

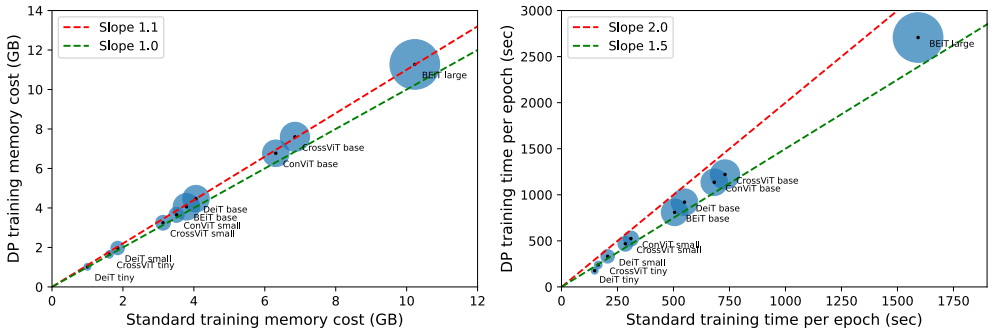

Figure 4: Memory (left) and speed (right) comparison of DP and non-DP training on CIFAR100 with convolutional ViTs. Note that CIFAR10 has an almost identical pattern.

6 Discussion

We have shown that DP training can be efficient for large CNNs and ViTs with convolutional layers. For example, in comparison to non-private training, we reduce the training time to $< 2\times$ and the memory overhead to $< 10\%$ for all vision models examined (up to 303.4 million parameters), including BEiT that achieves SOTA accuracy on CIFAR100 ($+15.6\%$ absolutely at $\epsilon = 1$). We have observed that for many tasks and large CNNs and ViTs, the memory overhead of DP training can be as low as less than 1%.

We emphasize that our DP training only improves the efficiency, not affecting the accuracy, and therefore is generally applicable, e.g. with SOTA data augmentations in [9]. With efficient training algorithms, we look forward to applying DP CNNs to generation tasks [19] , seq-to-seq learning [17], text classification [52], reinforcement learning [34], and multi-modal learning. Further reducing time complexity and prioritizing speed in DP training is another future direction.

In particular, our layerwise decision principle in (4.1) highlights the advantages of ghost clipping when $T = HW$ is small. This advocates the use of large kernel sizes in DP learning, as they shrink the hidden feature aggressively, and have been shown to be highly accurate on non-private tasks [20, 11, 36].

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 308–318, 2016.

[2] A. Ali, H. Touvron, M. Caron, P. Bojanowski, M. Douze, A. Joulin, I. Laptev, N. Neverova, G. Synnaeve, J. Verbeek, et al. Xcit: Cross-covariance image transformers. *Advances in neural information processing systems*, 34, 2021.

[3] H. Bao, L. Dong, and F. Wei. Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*, 2021.

[4] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.

[5] Z. Bu, S. Gopi, J. Kulkarni, Y. T. Lee, H. Shen, and U. Tantipongpipat. Fast and memory efficient differentially private-sgd via jl projections. *Advances in Neural Information Processing Systems*, 34, 2021.

[6] Z. Bu, H. Wang, Q. Long, and W. J. Su. On the convergence and calibration of deep learning with differential privacy. *arXiv preprint arXiv:2106.07830*, 2021.

[7] C.-F. R. Chen, Q. Fan, and R. Panda. Crossvit: Cross-attention multi-scale vision transformer for image classification. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 357–366, 2021.

[8] Z. Chen, L. Xie, J. Niu, X. Liu, L. Wei, and Q. Tian. Visformer: The vision-friendly transformer. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 589–598, 2021.

[9] S. De, L. Berrada, J. Hayes, S. L. Smith, and B. Balle. Unlocking high-accuracy differentially private image classification through scale. *arXiv preprint arXiv:2204.13650*, 2022.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[11] X. Ding, X. Zhang, Y. Zhou, J. Han, G. Ding, and J. Sun. Scaling up your kernels to 31x31: Revisiting large kernel design in cnns. *arXiv preprint arXiv:2203.06717*, 2022.

[12] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[13] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.

[14] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014.

[15] S. d'Ascoli, H. Touvron, M. L. Leavitt, A. S. Morcos, G. Biroli, and L. Sagun. Convit: Improving vision transformers with soft convolutional inductive biases. In *International Conference on Machine Learning*, pages 2286–2296. PMLR, 2021.

[16] K. Fukushima and S. Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pages 267–285. Springer, 1982.

[17] J. Gehring, M. Auli, D. Grangier, D. Yarats, and Y. N. Dauphin. Convolutional sequence to sequence learning. In *International Conference on Machine Learning*, pages 1243–1252. PMLR, 2017.

[18] I. Goodfellow. Efficient per-example gradient computations. *arXiv preprint arXiv:1510.01799*, 2015.

[19] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

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

[21] B. Heo, S. Yun, D. Han, S. Chun, J. Choe, and S. J. Oh. Rethinking spatial dimensions of vision transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11936–11945, 2021.

[22] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold, et al. Cnn architectures for large-scale audio classification. In *2017 ieee international conference on acoustics, speech and signal processing (icassp)*, pages 131–135. IEEE, 2017.

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

[24] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[25] H. Klause, A. Ziller, D. Rueckert, K. Hammernik, and G. Kaissis. Differentially private training of residual networks with scale normalisation. *arXiv preprint arXiv:2203.00324*, 2022.

[26] A. Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

[27] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[28] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[29] A. Kurakin, S. Chien, S. Song, R. Geambasu, A. Terzis, and A. Thakurta. Toward training at imagenet scale with differential privacy. *arXiv preprint arXiv:2201.12328*, 2022.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[31] J. Lee and D. Kifer. Scaling up differentially private deep learning with fast per-example gradient clipping. *arXiv preprint arXiv:2009.03106*, 2020.

[32] X. Li, F. Tramer, P. Liang, and T. Hashimoto. Large language models can be strong differentially private learners. *arXiv preprint arXiv:2110.05679*, 2021.

[33] H. Mehta, A. Thakurta, A. Kurakin, and A. Cutkosky. Large scale transfer learning for differentially private image classification. *arXiv preprint arXiv:2205.02973*, 2022.

[34] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[35] N. Papernot, A. Thakurta, S. Song, S. Chien, and U. Erlingsson. Tempered sigmoid activations for deep learning with differential privacy.

[36] C. Peng, X. Zhang, G. Yu, G. Luo, and J. Sun. Large kernel matters–improve semantic segmentation by global convolutional network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4353–4361, 2017.

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[38] G. Rochette, A. Manoel, and E. W. Tramel. Efficient per-example gradient computations in convolutional neural networks. *arXiv preprint arXiv:1912.06015*, 2019.

[39] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in neural information processing systems*, 27, 2014.

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

[41] P. Subramani, N. Vadivelu, and G. Kamath. Enabling fast differentially private sgd via just-in-time compilation and vectorization. *Advances in Neural Information Processing Systems*, 34, 2021.

[42] Torchvision. Pytorch vision description. https://pytorch.org/hub/pytorch_vision_vgg/ and https://pytorch.org/hub/pytorch_vision_resnet/.

[43] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou. Training data-efficient image transformers & distillation through attention. In *International Conference on Machine Learning*, pages 10347–10357. PMLR, 2021.

[44] H. Touvron, M. Cord, A. Sablayrolles, G. Synnaeve, and H. Jégou. Going deeper with image transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 32–42, 2021.

[45] F. Tramer and D. Boneh. Differentially private learning needs better features (or much more data). *arXiv preprint arXiv:2011.11660*, 2020.

[46] R. Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

[47] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[48] R. Yang, H. Ma, J. Wu, Y. Tang, X. Xiao, M. Zheng, and X. Li. Scalablevit: Rethinking the context-oriented generalization of vision transformer. *arXiv preprint arXiv:2203.10790*, 2022.

[49] A. Yousefpour, I. Shilov, A. Sablayrolles, D. Testuggine, K. Prasad, M. Malek, J. Nguyen, S. Ghosh, A. Bharadwaj, J. Zhao, G. Cormode, and I. Mironov. Opacus: User-friendly differential privacy library in PyTorch. *arXiv preprint arXiv:2109.12298*, 2021.

[50] D. Yu, H. Zhang, W. Chen, and T.-Y. Liu. Do not let privacy overbill utility: Gradient embedding perturbation for private learning. *arXiv preprint arXiv:2102.12677*, 2021.

[51] S. Zagoruyko and N. Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[52] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *Advances in neural information processing systems*, 28, 2015.

[53] Z. Zhang, H. Zhang, L. Zhao, T. Chen, S. Arik, and T. Pfister. Nested hierarchical transformer: Towards accurate, data-efficient and interpretable visual understanding. 2022.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [N/A]
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] No theoretical results.
   (b) Did you include complete proofs of all theoretical results? [N/A] No theoretical results.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Experiment details are included in the appendix. We will release a URL for the experiments.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See appendix.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] We state in the experiment section that we use a single Tesla V100 GPU.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] We cite clearly the codebases such as TorchVision, Pytorch Image Models, Opacus, etc. We also cite the datasets properly.
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] All datasets are publically available.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A] All datasets are publically available.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]