Local Differential Privacy for Deep Learning

M.A.P. Chamikara[a,b,*], P. Bertok[a], I. Khalil[a], D. Liu[b], S. Camtepe[b]

[a] *School of Science, RMIT University, Australia*
[b] *CSIRO Data61, Australia*

Abstract

Deep learning (DL) is a promising area of machine learning which is becoming popular due to its remarkable accuracy when trained with a massive amount of data. Often, these datasets are highly sensitive crowd-sourced data such as medical data, financial data, or image data, and the DL models trained on these data tend to leak privacy. We propose a new local differentially private (LDP) algorithm (named LATENT) which redesigns the training process in a way that a data owner can add a randomization layer before data leave data owners' devices and reach to a potentially untrusted machine learning service. This way LATENT prevents privacy leaks of DL models, e.g., due to membership inference and memorizing model attacks, while providing excellent accuracy. By not requiring a trusted party, LATENT can be more practical for cloud-based machine learning services in comparison to existing differentially private approaches. Our experimental evaluation of LATENT on convolutional deep neural networks demonstrates excellent accuracy (e.g. 91%- 96%) with high model quality even under very low privacy budgets (e.g. $\epsilon = 0.5$), outperforming existing differentially private approaches for deep learning.

Keywords: Data privacy, deep learning, differential privacy, local differential privacy

1. Introduction

Compared to traditional machine learning approaches, deep Learning (DL) shows remarkable success over complex problems such as image classification, natural language processing, and speech recognition. DL models are often trained on sensitive crowd-sourced data such as personal images, health records, and financial records. When DL models are trained on massive sensitive databases, they tend to expose private information [1, 2]. With the advancement of distributed cloud-based machine learning environments such as the ones offered by Google and Amazon [3, 4], more users may become

*Corresponding author
Email address: pathumchamikara.mahawagaarachchige@rmit.edu.au (M.A.P. Chamikara)

vulnerable to such attacks. Trusting these environments, users may feed their data to train the models and obtain white-box or black-box access to these models without being concerned about the actual training process. However, an adversary can easily implement malicious algorithms and offer them as part of these training processes. Malicious algorithms may memorize the sensitive user information as part of the trained models. Adversaries can later extract and approximate the memorized information, and thereby obtain information about the users and breach their privacy [5]. Privacy inference attacks, such as membership inference show the vulnerability of deep learning models trained on sensitive data even when they are released as black box models [2]. Another example that shows the weakness of trained ML models is model inversion attacks that recover images from a facial recognition system [6]. It is essential that machine learning as a service employs sufficient privacy-preserving mechanisms to limit privacy leaks of trained DL models.

In this paper, we examine the privacy issues of deep learning and develop a robust privacy preserving mechanism to control privacy leaks in deep learning. The existing benchmark privacy-preserving approaches for deep learning are based on global differential privacy (GDP) [7, 1]. GDP [1] and LDP [8] are the two main classes of differential privacy (DP). DP constitutes a robust framework guaranteeing strong levels of privacy [9]. In GDP, a trusted curator will employ calibrated noise to provide differential privacy as shown in Figure 1 [10, 11]. In LDP, owners will perturb their data before releasing them, to provide better privacy without trusting any third party as depicted in Figure 1 [11]. Existing GDP methods are not suitable for practical DL services such as those offered by Google, as they require a trusted curator. In such a scenario, the GDP algorithm should reside in the server and the original data need to be uploaded to the server for training, which poses a threat to privacy e.g. an adversary performing membership inference attacks using Google's ML models or providing a Google hosted malicious ML service for memorizing model attacks. Moreover, DL algorithms are inherently computationally complex, and privacy-preserving solutions on DL models also tend to be complex and need high computational processing power. Consequently, GDP algorithms are preferred to run on high-performance computers, and resource constrained data owners can not use them in untrusted environments. Furthermore, noise calibration of GDP methods, such as Laplacian and Gaussian mechanisms for ML models can be complex, indefinite and produce less accurate results or entail a higher level of privacy leak [7, 1].

Our contribution is an LDP mechanism for limiting the privacy leaks of convolutional neural network (CNN) models that are released as black box models. The proposed algorithm (named as LA-

TENT) employs the properties of randomized response [12], a popular survey technique that satisfies local differential privacy. The LDP setting of LATENT allows privacy-preserving communication between several parties which is not possible with existing GDP methods for deep learning. As the LDP approach of LATENT enables control of the privacy budget before the perturbation process, accuracy can be effectively tuned independently. In other words, LATENT reduces the impact of the privacy budget (ϵ) on accuracy, and this leads to significantly higher privacy and accuracy than existing solutions display. Compared to current GDP methods for deep learning, LATENT provides excellent accuracy (above 90%) under extreme cases of privacy budgets (e.g. $\epsilon = 0.5$) that ensure minimum leak. Our experiments clearly show that a general purpose computer is sufficient to perform the required computations efficiently and reliably at the data owner's end. Accordingly, LATENT can be a more practical and robust tool to limit the privacy leak of deep learning models than existing methods.

The rest of the paper is organized as follows. The underlying concepts used in LATENT are presented in Section 2. Section 3 explains the steps of the differentially private mechanism for deep learning. The results of LATENT are discussed in Section 4. Section 5 provides a summary of existing related work. The paper is concluded in Section 6.

2. Background

In this section, we provide brief descriptions of the underlying concepts of LATENT. The section includes brief summaries of basic principles related to "Differential Privacy" and "Deep learning" which are used in LATENT.

2.1. Differential Privacy

Differential privacy (DP) is a privacy model that is known to render maximum privacy by minimizing the chance of individual record identification [11]. In principle, DP defines the bounds to how much information can be revealed to a third party/adversary about someone's data being present in a particular database. Conventionally ϵ (epsilon) and δ (delta) are used to denote these bounds, which decide the level of privacy rendered by a randomized privacy preserving algorithm (M) over a particular database (D).

2.1.1. privacy budget/privacy loss (ϵ)

ϵ is called the privacy budget that provides an insight into the privacy loss of a DP algorithm. The higher the value of ϵ, the higher the privacy loss.

2.1.2. probability to fail/probability of error (δ)

δ is the parameter that accounts for "bad events" that might result in high privacy loss; δ is the probability of the output revealing the identity of a particular individual, which can happen $\delta \times n$ times where n is the number of records. To minimize the risk of privacy loss, $\delta \times n$ has to be maintained at a low value. For example, the probability of a bad event is 1% when $\delta = 1/100 \times n$.

2.1.3. definition of differential privacy

Let's take two adjacent datasets of D, x and y, where y differs from x only by one person. Then M satisfies (ϵ, δ)-differential privacy if it holds Equation (1).

Definition 1. *A randomized algorithm M with domain $N^{|X|}$ and range R: is (ϵ, δ)-differentially private for $\delta \geq 0$ if for every adjacent x, $y \in N^{|X|}$ and for any subset $S \subseteq R$*

$$Pr[(M(x) \in S)] \leq \exp(\epsilon) Pr[(M(y) \in S)] + \delta \tag{1}$$

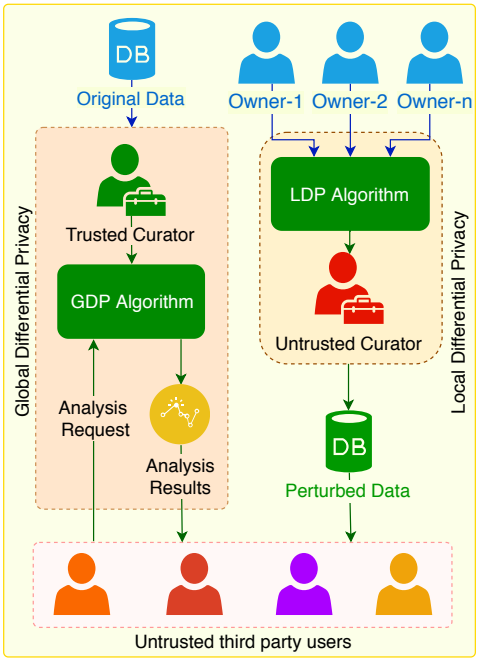

Figure 1: Global vs. Local differential privacy

2.2. Global vs. Local Differential Privacy

As depicted in Figure 1, global differential privacy (GDP) and local differential privacy (LDP) are two approaches that can be used by randomized algorithms to achieve differential privacy. In the GDP

setting, there is a trusted curator who applies carefully calibrated random noise to the real values returned for a particular query. The most frequently used noise generation processes for GDP include Laplace mechanism and Gaussian mechanism [9]. The GDP setting is also called the trusted curator model [13]. A randomized algorithm, M provides (ϵ, δ)-global differential privacy if it holds Equation (1). LDP needs no trusted third party, hence it is also called the untrusted curator model [11]. With LDP, the data is randomized before the curator can access it. LDP can also be used by a trusted party to randomize all records in a database at once. The right-hand column of Figure 1 represents the LDP setting. LDP algorithms may often produce too noisy data, as noise is applied commonly to achieve individual data privacy. LDP is considered to be a strong and rigorous notion of privacy that provides plausible deniability. Due to the above properties, LDP is deemed to be a state-of-the-art approach for privacy-preserving data collection and distribution. A randomized algorithm A provides ϵ-local differential privacy if it holds Equation (2) [14].

Definition 2. *A randomized algorithm A satisfies ϵ-local differential privacy if for all pairs of inputs v_1 and v_2 and for all $Q \subseteq Range(A)$, and for ($\epsilon \geq 0$), A holds Equation* (2)*. $Range(A)$ is the set of all possible outputs of the randomized algorithm A.*

$$Pr[A(v_1) \in Q] \leq \exp(\epsilon) Pr[A(v_2) \in Q] \tag{2}$$

2.3. Randomized Response

Randomized response is a survey technique to eliminate evasive answer bias by randomizing the responses to a survey question with the answer "yes" or "no" [15]. An answer is randomized by flipping two independent, unbiased coins. The answer is truthful if the first coin comes up "heads", else, the second coin is flipped, and the answer is "yes" if "heads", "no" if "tails". Assume that the coins are biased and the probability of a coin turning up heads is p. It has been shown that randomized response provides ϵ- differential privacy when $p = e^{\epsilon}/(1 + e^{\epsilon})$ [11].

2.4. Sensitivity, Privacy Budget (ϵ), and determination of the probability (p) of randomization

To quantify the probability of randomization (p) of an LDP process that is based on transferring bit strings, we can use the method employed by RAPPOR (Randomized Aggregatable Privacy-Preserving Ordinal Response), which is an LDP algorithm proposed by Google [14]. RAPPOR is motivated by the problem of estimating a client-side distribution of string values drawn from a discrete data dictionary.

One application of RAPPOR is to track the distribution of users' browser configuration strings in the Chrome web browser.

Sensitivity is defined as the maximum influence that a single individual can have on the result of a numeric query. Consider a function f, the sensitivity (Δf) of f can be given as in Equation (3) where x and y are two neighboring databases and $\|.\|_1$ represents the $L1$ norm of a vector [16].

$$\Delta f = max\{\|f(x) - f(y)\|_1\} \tag{3}$$

Since RAPPOR is an LDP algorithm, it considers x and y to be a pair of adjacent inputs in RAPPOR's definition of global sensitivity. In RAPPOR any input is a vector of d bits, and each d-bit vector contains $d-1$ zeros and 1 one, so the maximum difference, Δf (the sensitivity), is 2 bits. In other words, for the underlying data representation of RAPPOR, f has a sensitivity of 2. RAPPOR is an LDP algorithm when the probability (p) of randomization follows Equation (4), where ϵ is the privacy budget offered by the LDP process [14, 17].

$$p = \frac{e^{\frac{\epsilon}{\Delta f}}}{1 + e^{\frac{\epsilon}{\Delta f}}} = \frac{e^{\frac{\epsilon}{2}}}{1 + e^{\frac{\epsilon}{2}}} \tag{4}$$

2.5. Properties of Differential Privacy

Postprocessing invariance/robustness, quantifiability, and composition are three of the essential characteristics of differential privacy [18]. Although additional computations are carried out on the outcome of a differentially private algorithm, they do not weaken the privacy guarantee. So, the results of additional computations on ϵ-DP outcome will still be ϵ-DP. This property of DP is called the postprocessing invariance/robustness. Quantifiability is the ability of DP scenarios to provide transparency in calculating the precise amount of perturbation applied by a particular randomization process. Thus, the user of a particular DP algorithm knows the level of privacy provided by the data/results released after the perturbation. Composition is the degradation of privacy when multiple differentially private algorithms are performed on the same or overlapping datasets [18]. According to DP definitions, when two DP algorithms; ϵ_1-DP and ϵ_2-DP are applied on the same or overlapping datasets, the union of the results is equal to $(\epsilon_1 + \epsilon_2)$-DP [18]. The more DP algorithms are applied to the same data, the more privacy loss is accumulated. Depending on the process of synthesis, DP algorithms can be categorized into the two types; basic algorithms or derived algorithms [19]. Differential privacy is self-contained in basic algorithms while the derived algorithms are derived from

existing methods by applying the theories of composition and postprocessing invariance.

2.6. Deep Learning using convolutional neural networks

A CNN is commonly trained to recognize essential features of images. A CNN uses a collection of layers named convolution layers with large receptive fields. A sequence of steps through a stack of convolution layers is followed by an intermediate functionality called pooling to reduce the dimensions from the previous layer to the next layer. The final pooled output which is produced from the last convolution layer is flattened to produce a sizeable 1-D vector [20]. Then a fully connected artificial neural network (ANN) is trained using these input vectors to generate predictions on the inputs (images). An ANN is more or less a connected network of processing modules called neurons, each producing a sequence of real-valued activations.

Overfitting is the situation where the training accuracy is significantly higher than the testing accuracy [20, 21]. Underfitting is the situation where the testing accuracy is significantly higher than the training accuracy. A model with better quality is considered to avoid these problems. Regularization, image augmentation, and hyperparameter tuning are three of the commonly used concepts to avoid these problems and improve the performance and robustness of neural networks [20, 21]. Regularization is the process of applying any modification to a learning algorithm to reduce the generalization error. Regularization can be achieved using dropouts where a certain percentage of neurons are randomly dropped in each epoch (training cycle) to avoid overfitting. Image augmentation is a data preparation technique which uses the existing input images in the training dataset and manipulates them to create many altered versions of the same input using different transformation methods such as reflection, sheer, and rotation. This technique allows the ANN to learn a wider variety of inputs to make the trained model more generalizable with high robustness [22]. In hyperparameter tuning the inputs to hyperparameters such as the percentage dropout, the batch size, the activation function, the number of neurons, the number of epochs, and the optimizer are changed under different training phases to identify the best case study which returns the best results [20, 21].

The percentage dropout is a technique to ignore a percentage of randomly selected neurons being trained during a single cycle of training [23]. The batch size is the number of training examples that are going to be propagated in one forward/backward pass [21]. Activation functions define the output of a particular neuron given an input which is the sum of products of all the inputs and the corresponding weights to introduces non-linear properties to the network [20]. A neuron (also called a node) is the primary component of an artificial neural network. Too many nodes often make the

learning process inefficient hence, eliminating unnecessary/redundant nodes can be significant. An insufficient number of nodes may lead to reduced training accuracy [20]. A single pass in which the entire dataset is introduced forward and backward through the neural network is called an epoch [20]. An optimizer (or an optimization algorithm) is used to update the model parameters such as weights and bias values [20]

3. Our Approach: LATENT

This section discusses the differentially private mechanism employed in LATENT for deep learning. LATENT can be classified as a derived LDP algorithm which is based on the randomized response technique. LATENT uses the two properties of differential privacy: postprocessing invariance and composition to generate a CNN model with differential privacy. LATENT uses regularization, image augmentation, and hyperparameter tuning to optimize its performance under noisy input conditions resulted in the randomization process. We implemented and tested LATENT on convolutional neural networks using the Python Keras neural networks API, which runs on top of TensorFlow dataflow engine developed by Google [3, 24]. Keras provides a high-level neural networks API which is designed primarily for fast experimentation.

3.1. Introduction of the intermediate layer (LATENT) to inject differential privacy to the CNN architecture

As shown in Figure 2, we divide the structure of a convolutional neural network into two main modules and introduce an intermediate module of randomization to the CNN structure. Recall (described in Section 2.6) that in CNN, the input features are initially subjected to dimensionality reduction, using a collection of convolutional layers and pooling layers. The output of the final pooling layers is flattened into a single 1-d array before feeding it to a fully connected artificial neural network. We call this part of a CNN as the convolutional module, and we name the ANN component of the CNN as the ANN module. We introduce the randomization layer which is named as LATENT in between the convolutional module and the ANN module as shown in Figure 2. In the proposed architecture we use the convolutional module only to generate the 1-d flattened output that corresponds to a particular image input. This flattened output is simply a one-dimensional column vector of float values (real-valued numbers).

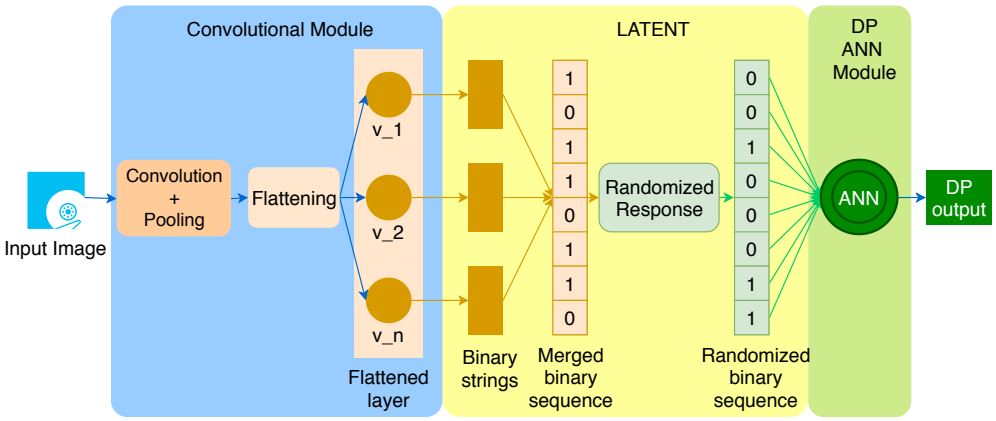

Figure 2: CNN architecture with the LATENT randomization layer

3.1.1. Apply z-score normalization on the input values to LATENT

LATENT converts the input values to binary values before the application of randomization. The inputs can have different ranges. Conversion of large values or small fractions into binary can involve a large number of bits. This can introduce an inconsistent level of complexity to the algorithm. To avoid this complexity, we apply z-score normalization to the values of the 1-d vector coming from the flattening layer.

3.1.2. Define the bounds (lengths of the segments) for the binary conversion

The length of the bit pattern establishes the range of a particular z-score normalized input. The upper bound and the lower bound of a specific input needs to be initially estimated. Figure 3 represents the arrangement of bits of the binary conversion of a z-score normalized input. As shown in the figure, there are three primary segments of the binary string. The sign bit will represent the sign of the input, 1 for negative and 0 for positive. The other two parts are for the whole number and the fraction part of an input number respectively. Selection of the number of bits for the whole number depends on the maximum value of the whole number that needs to be represented. Due to z-score normalization, the number of bits necessary to represent the whole number is small. Selection of the number of bits for the fraction depends on the precision (how close is the binary fraction's decimal value to the input's fraction value). For more precision, a higher number of bits needs to be used for the fraction.

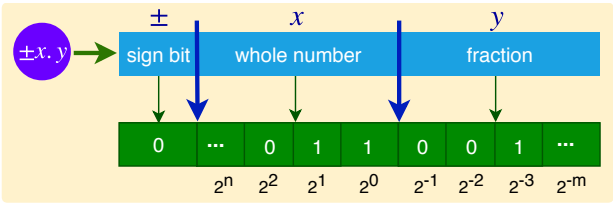

Figure 3: Direct mapping of a float/integer to binary

3.1.3. Convert each value of the flattened layer to binary using the bounds

After determining the length of the components of binary strings of the inputs, the inputs can be mapped as shown in Figure 3. The figure shows the direct mapping of an integer/float value to its binary representation. The binary representation can be generated according to Equation (5), where n and m are the numbers of binary digits of the whole number and the fraction respectively, x represents the original input value where $x \in \mathbb{R}$, and $g(i)$ represents the i^{th} bit of the binary string where the least significant bit is represented when $k = -m$. The sign bit is 1 for negative values and 0 for positive values. The sign bit is assigned to the most significant bit of the binary string.

$$g(i) = \left(\left\lfloor 2^{-k} |x| \right\rfloor \mod 2 \right)_{k=-m}^{n} \text{ where, } i = k + m \tag{5}$$

3.1.4. Merge the binary strings to reduce the privacy loss

We merge all the binary strings into one long binary string to avoid privacy loss due to the composition property of differential privacy. If we conducted the randomization on each binary string corresponding to the flattened 1-d vector separately, it would add up the privacy budgets of all the randomization steps. If r binary strings were randomized, the resulting privacy loss of the final randomization would be $r \times \epsilon$. As LATENT conducts the randomization on a particular merged binary string at once, we can maintain the privacy loss at the input value of ϵ.

3.1.5. Define the probability of randomization (p) in terms of ϵ

The probability of randomization (p) is calculated in terms of the privacy budget (ϵ) before the randomization of the merged binary string. Recall that in the randomized response technique (as described in Section 2.4) used in RAPPOR, when the difference of the number of bits of two neighboring inputs is d, the sensitivity becomes d. In the case of LATENT, the length of a binary string is $l = (n + m + 1)$ which makes the length of the merged binary string equal to $l \times r$ where r is the number of outputs of the flattening layer of the convolutional module. According to our method of binary conversion, two consecutive inputs can differ by at most of $l \times r$ bits. Consequently, according

to RAPPOR, we have a sensitivity of $l \times r$. Now, we can represent the probability of randomization according to Equation (6). We can notice that merging the binary strings together increases sensitivity, hence, increasing the amount of randomization necessary.

$$p = \frac{e^{\epsilon/rl}}{1 + e^{\epsilon/rl}} \tag{6}$$

3.1.6. Conduct randomized response on the bits of the merged binary strings

Each bit in the merged binary string is subjected to randomized response with a probability of randomization equal to p, given in Equation (6). The higher the p is, the lower the randomization of the binary string will be. According to Equation (6), higher ϵ values and lower $l \times r$ values will result in higher p values. In LATENT, $l \times r$ is a considerably larger value than ϵ, p is often a smaller value. Thus, for smaller single digit values of ϵ, LATENT applies maximum randomization over the binary strings.

3.1.7. Generate a differentially private classification model using the ANN module

After LATENT randomizes merged binary strings it feeds the randomized binary strings to the ANN module of the convolutional network. The ANN module is then trained on the randomized binary strings to generate a differentially private ANN model. We improve the performance of the differentially private model using regularization, image augmentation, and hyperparameter tuning.

3.2. Algorithm for generating a differentially private CNN

Algorithm 1 shows the steps of LATENT in producing a differentially private output. It provides the precise sequence of summarized steps explained under Section 3.1 in applying differential privacy to the CNN architecture.

Algorithm 1: Differentially private CNN model generation

Input:
- $\{x_1, \ldots, x_j\}$ ← examples
- ϵ ← privacy budget
- n ← number of bits for the whole number of the binary representation
- m ← number of bits for the fraction of the binary representation

Output: $DPCNN$ ← differentially private CNN model

1 define the convolutional module (CNM) as explained in Section 3.1;

2 declare, $l = (m + n + 1)$;

3 feed $\{x_1, \ldots, x_j\}$ to the CNM and generate the sequence of 1-d feature arrays $\{d_1, \ldots, d_j\}$;

4 convert each field (x) of d_q (where, $q = 1, \ldots, j$) to binary using,
$g(i) = \left(\lfloor 2^{-k} |x| \rfloor \mod 2 \right)_{k=-m}^{n}$ where, $i = k + m$;

5 generate array $\{b_1, \ldots, b_j\}$ of the merged binary arrays for the elements in $\{d_1, \ldots, d_j\}$;

6 determine the length (r) of a single element of $\{d_1, \ldots, d_j\}$;

7 calculate randomization probability, $p = \frac{e^{\epsilon/rl}}{1+e^{\epsilon/rl}}$;

8 randomize each element of $\{b_1, \ldots, b_j\}$ using LATENT with probability p to generate $\{pb_1, \ldots, pb_j\}$;

9 train the ANN module of the CNN using $\{pb_1, \ldots, pb_j\}$;

10 optimize the ANN module using regularization, image augmentation and/or hyperparameter tuning;

11 release the DPCNN;

3.3. Error of estimation

Let's consider the randomization of a single element x in the database to generate a perturbed value x^p where both x and x^p are base 10 values. Assume that the proportion of 1s in the binary representation of x^p is t and the estimated proportion of 1s in the binary representation of x is y. Now, t can be represented using Equation (7) where p is the probability of randomization (given in Equation (6)).

$$t = y \times p + \frac{(1-p)}{2} \tag{7}$$

Therefore,

$$2t = 2y \times p + (1-p) \tag{8}$$

$$y = \frac{2t - (1-p)}{2p} \tag{9}$$

Since, y, provides an estimate to the proportion of $1s$ available in x, if the number of bits in x or x^p is l, x will have $l \times y$ number of estimated 1s. x can be one of $\binom{l}{k}$ possibilities where, $k = round(l \times y)$.

Consider a function $F(l, k, i)$ which gives the i^{th} number in increasing order that has l binary bits of which k bits are 1s. Let i range from 0 to $\binom{l}{k} - 1$. There are $\binom{l-1}{k}$ that start with a 0 in the most significant bit and $\binom{l-1}{k-1}$ that start with a 1. So, we can represent $F(l, k, i)$ using the recursive function as shown in Equation (10).

$$F(l,k,i) = \begin{cases} 0 & k = 0 \\ 2^{k-1} & l = k \\ F(l-1,k,i) & i < \binom{l-1}{k} \\ 2^{l-1} + F(l-1,k-1,i-\binom{l-1}{k}) & i \geq \binom{l-1}{k} \end{cases} \tag{10}$$

We can now determine an error of estimation of x using x^p according to Equation (15), provided that the adversary knows the process of randomization used in the perturbation. Based on Equation (9) and Equation (15), it can be noted that the error of estimating x using x^p can be substantial. This indicates the depth of perturbation applied to a particular input by LATENT.

$$A = E(x^p - x) = \left\{ x^p - (F(l,k,i))_{i=1}^{\binom{l}{k}} \right\}_{i=1}^{\binom{l}{k}} \tag{11}$$

$$A' = abs\left\{E(x^p - x)\right\}_{i=1}^{\binom{l}{k}} \tag{12}$$

$$B = min(A') \tag{13}$$

$$C = \left\{ i | a'_i = B \text{ where } a'_i \in A' \right\} \tag{14}$$

$$\widehat{E} = \left\{ a_i | i \in C \text{ and } a_i \in A \right\} \tag{15}$$

3.4. The LDP settings for LATENT

Since the randomization takes place after the convolutional module, we push the convolutional module and the LATENT module to the data owner's end as shown in Figure 4. The DP ANN model is trained at the untrusted curator's end which can be a cloud computer or any high-performance computing server. The model release will involve the release of only the trained DP ANN module which can be used for testing by any third party. In the proposed setting, the convolutional module is

not trained for features, leaving a minimum computational burden on a particular data owner.

The proposed component distribution of the CNN architecture, which moves the convolutional module to the data owner, produces additional privacy even before the randomization, as the output of the CNN module is a dimension-reduced 1-d vector. Additionally, in the big data context where millions of data owners communicate with the server, our CNN model distribution can provide additional flexibility and efficiency in data processing, leaving the ANN module to train on the already dimension-reduced data.

However, we can push the whole DP CNN architecture to a single machine where we keep a central repository that is maintained by a trusted curator. Then we can apply the CNN with LATENT on the dataset at the trusted curator's end where the model will be released with the whole architecture of the CNN (Convolutional module (untrained)+LATENT+DP ANN module).

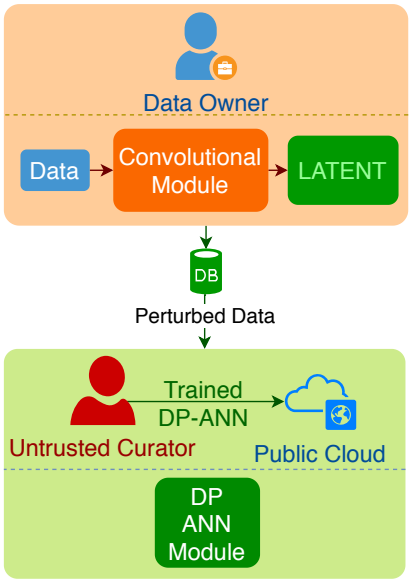

Figure 4: LDP configurations of LATENT

4. Results and Discussion

In this section, we discuss the experiments, experimental configurations, and their results. We tested our method using the MNIST dataset [25] and the CIFAR-10 dataset [1] which are considered to be the benchmark datasets to train and test deep learning (CNN) algorithms. We specifically selected MNIST and CIFAR-10 for the experiments as they have been used in recent works on deep learning with differential privacy [7, 1]. MNIST is famous for generating good accuracy in deep learning, whereas CIFAR-10 is a complex dataset and is difficult for training. These complementary properties

of MNIST and CIFAR-10 provide a balanced experimental setup to test the performance of a specific deep learning scenario. We conducted all experiments on an HPC cluster (SUSE Linux Enterprise Server 12 SP3) with 112 Dual Xeon 14-core E5-2690 v4 Compute Nodes each with 256 GB of RAM, FDR10 InfiniBand interconnect, and 4 NVidia Tesla P100 (SXM2). The computational burden of LATENT on resource-constrained data owners was evaluated using a general purpose Intel Core i5 computer. A comprehensive specification of the corresponding computer is provided in Section 4.2.

4.1. Experimental setup

First, we created suitable baseline CNN models for each dataset. The baseline models include CNN without differential privacy (NPCNN) and a differentially private version of the same configuration (DPCNN). Figure 5 and Figure 6 show the baseline CNN architectures defined for the MNIST dataset and the CIFAR-10 datasets respectively. In the figures, the left-hand side models represent the NPCNN. The right-hand side models are the differentially private versions (DPCNN) of the corresponding left-hand side models. First we tested the accuracy of the NPCNN models, then we tested the DPCNN models' performance relative to NPCNN models, and conducted hyperparameter tuning and image augmentation on the DPCNN models to improve performance. Finally, the results of the best DPCNN models were chosen to compare the results with other existing differentially private methods for deep learning.

4.1.1. Datasets and CNN model information

This section provides information about the datasets and the architectures of the corresponding CNN models used in the experiments. The architecture of a CNN needs to be custom configured, as the performance depends on the characteristics of the input dataset. The model quality of the trained ANN depends on the correct configuration of its network architecture [20]. As explained below, we declared suitable CNN architectures for the two datasets separately, because CIFAR-10 is a more complex dataset than MNIST.

4.1.1.1. MNIST. The MNIST dataset is composed of 70,000 grayscale handwritten digits, where 60,000 examples are used for training, and a 10,000 are used for testing. Each image has a resolution of 28x28. The digits have been size-normalized and centered in a fixed-size image [25]. Figure 5 depicts the CNN network architecture used in the baseline models for the MNIST dataset. The figure shows the sequence of the layers of the network architectures of the baseline models. The network accepts 28×28 input images. The convolutional layer (layer 2) uses 32, 3×3 filters with stride 1

followed by a second convolutional layer (layer 3) which uses 64, 3×3 filters with stride 1. Both layer 2 and 3 use ReLU as the activation function. The output of layer 3 is subjected to a max pooling layer with 2×2 max pools. Thus, the max pooling layer outputs a 12×12×64 tensor for each image. Next, the output of the max pooling layer is subjected to a dropout of 25% (layer 5) and flattened (layer 6) to a 1-d vector of size 9216. The output of the flattening layer is fed into a fully connected layer (layer 7) with 128 neurons with ReLU activation function, followed by a dropout of 50%. The output of the dropout layer is finally fed into a fully connected layer with 10 neurons which produces the final output of the CNN network. This model (NPCNN) achieves 99.25% training and 98.16% testing accuracies after 12 epochs of training with a batch size of 128 using the Adadelta optimization algorithm.

4.1.1.2. DPCNN for MNIST. The DPCNN has an additional layer: LATENT (layer number 6, colored in yellow) of randomization in between the convolutional module and the ANN module. The green square represents the convolutional module, and the orange square represents the ANN module. If the LATENT layer uses 10 bits to represent one element (one output of the flattening layer) coming from the fattening layer, the length of the randomized bit string generated by the LATENT layer is equal to 10 times the number of outputs of the flattening layer. In this case, the length of the randomized bit string will be equal to $9216 \times 10 = 92160$. The green arrows are used to indicate that the same configuration of the corresponding layer is available in the DPCNN. The red cross is used to indicate that the corresponding layer was omitted from the DPCNN. As shown in the figure, we do not use dropouts in the convolutional module of the DPCNN, as the convolutional module is not trained for the input features, which is explained in Section 3.1. In the DPCNN experiments with the MNIST dataset, we maintained a fixed size of 10 bits to represent each output of the flattening layer. The 10 bits are composed of 4 bits for the whole number, 5 bits for the fraction and 1 bit for the sign.

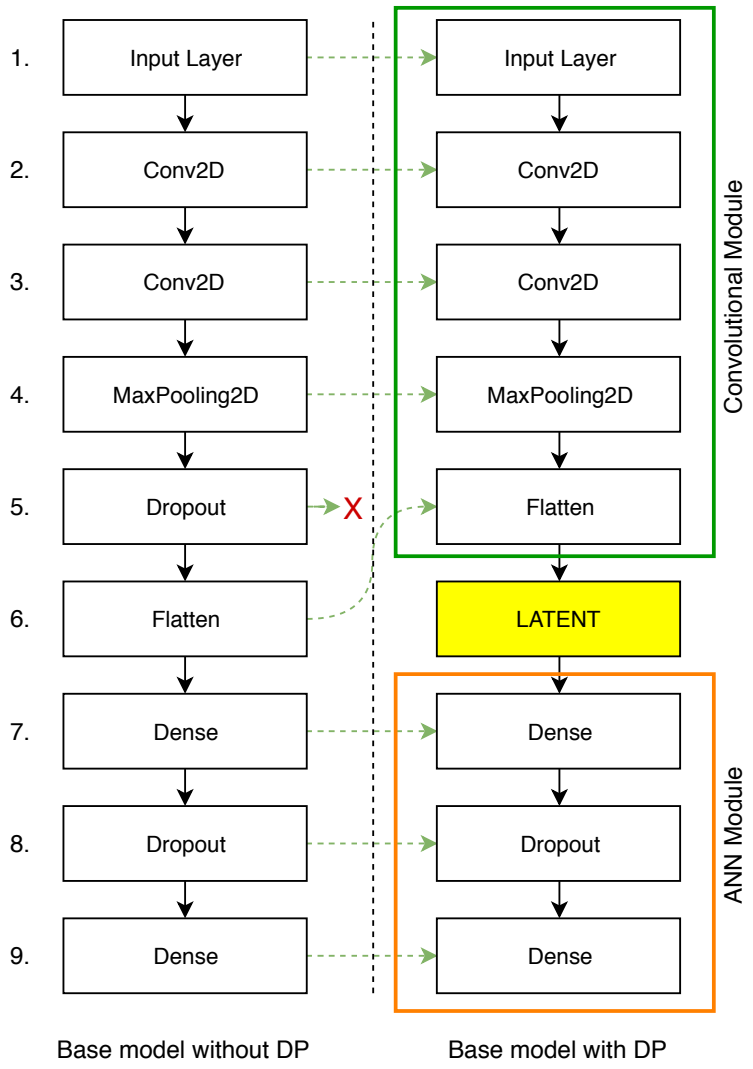

Figure 5: Architectural differences between the non-private (NPCNN) and differentially private (DPCNN) baseline models for the handwriting recognition dataset (MNIST dataset)

4.1.1.3. CIFAR-10. The CIFAR-10 dataset consists of 60000 color images and 10 classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck), with 6000 images per class. There are 50000 training images and 10000 testing images. Each image has a resolution of 32x32 [1]. Figure 6 depicts the CNN architecture used in the baseline models for the CIFAR-10 dataset. As CIFAR-10 is a complex dataset compared to MNIST, for CIFAR-10 we considered a more complex CNN architecture which involves more layers and more neurons. The network accepts 32×32, 3 channel input images. The convolutional layer (layer 2) uses 32, 3×3 filters with stride 1 followed by a second convolutional layer (layer 3) which uses 32, 3×3 filters with stride 1. Both layers 2 and 3 use ReLU as the activation

function. The output of layer 3 is subjected to a max pooling layer with 2×2 max pools. The output of layer 4 is fed to a dropout layer (layer 5) with 25% dropout. The output of layer 5 was subjected to two other convolutional layers (layer 6 and 7) which use 64, 3×3 filters. The output of layer 6 was next introduced with a max pooling layer (layer 8) with 2×2 max pools. Thus, the max pooling layer outputs a 6×6×64 tensor for each image. Next, the output of the max pooling layer is subjected to a dropout of 25% (layer 9), and flattened (layer 10) to a 1-d vector of size 2304. The output of the flattening layer is fed into a fully connected layer (layer 11) with 512 neurons with ReLU activation function, followed by a dropout of 50%. The output of the dropout layer is finally fed into a fully connected layer with 10 neurons which produces the final output of the CNN network. This model (NPCNN) achieves 73.32% training and 78.75% testing accuracies after 100 epochs of training for a batch size of 32 using the Adadelta optimization algorithm.

4.1.1.4. DPCNN for CIFAR-10. The right-hand side figure of Figure 6 represents the DPCNN for the CIFAR-10 dataset. The DPCNN model creation follows the same approach explained under the model creation process for the MNIST dataset. We do not use dropouts in the convolutional module for reasons explained in Section 3.1. In the experiments with the DPCNN model for the CIFAR-10 dataset, we maintained a fixed size of 10 bits to represent each output of the flattening layer. The 10 bits are composed of 2 bits for the whole number, 7 bits for the fraction and 1 bit for the sign.

Table 1: List of values applied for each hyperparameter in the test case generation process of hyperparameter tuning

Hyperparameter	**Hyperparameter settings for MNIST**	**Hyperparameter settings for CIFAR-10**
percentage dropout	layer 8 => {20%, 40%, 50%}	layer 11 => {20%, 40%, 50%}
batch size	{ 128, 256, 512}	{400, 500, 600}
activation function	layer 7=> {relu, tanh, sigmoid}	layer 10=> {relu, tanh, sigmoid}
the number of neurons	layer 7=> {64, 128, 256, 512}	layer 10=> {256, 512, 1024}
number of epochs	{50, 100, 150}	{100, 200, 300}
optimizer	{SGD, Adadelta, Adam}	{SGD, Adadelta, Adam}

4.1.2. Hyperparameter Tuning and Regularization

As explained in Section 3.1, we conducted the training only on the ANN module. Therefore, we apply hyperparameter tuning and regularization only on the ANN module. Given the number of possible values for each hyperparameter, the number of test cases can become large, and it may entail a substantial computational cost with exponential time. Therefore, it can be imperative to use insights such as used by [1] to minimize the number of hyperparameter settings that need to be tested. Since LATENT does not change the internal parameters of the ANN module, the architectural modifications necessary can be thought of as an independent procedure that can be common in any ANN training process. Consequently, we tested different combinations for percentage dropouts, batch sizes, activation functions, number of neurons, optimizers, number of epochs, for the hyperparameter tuning process as given in Table 1.

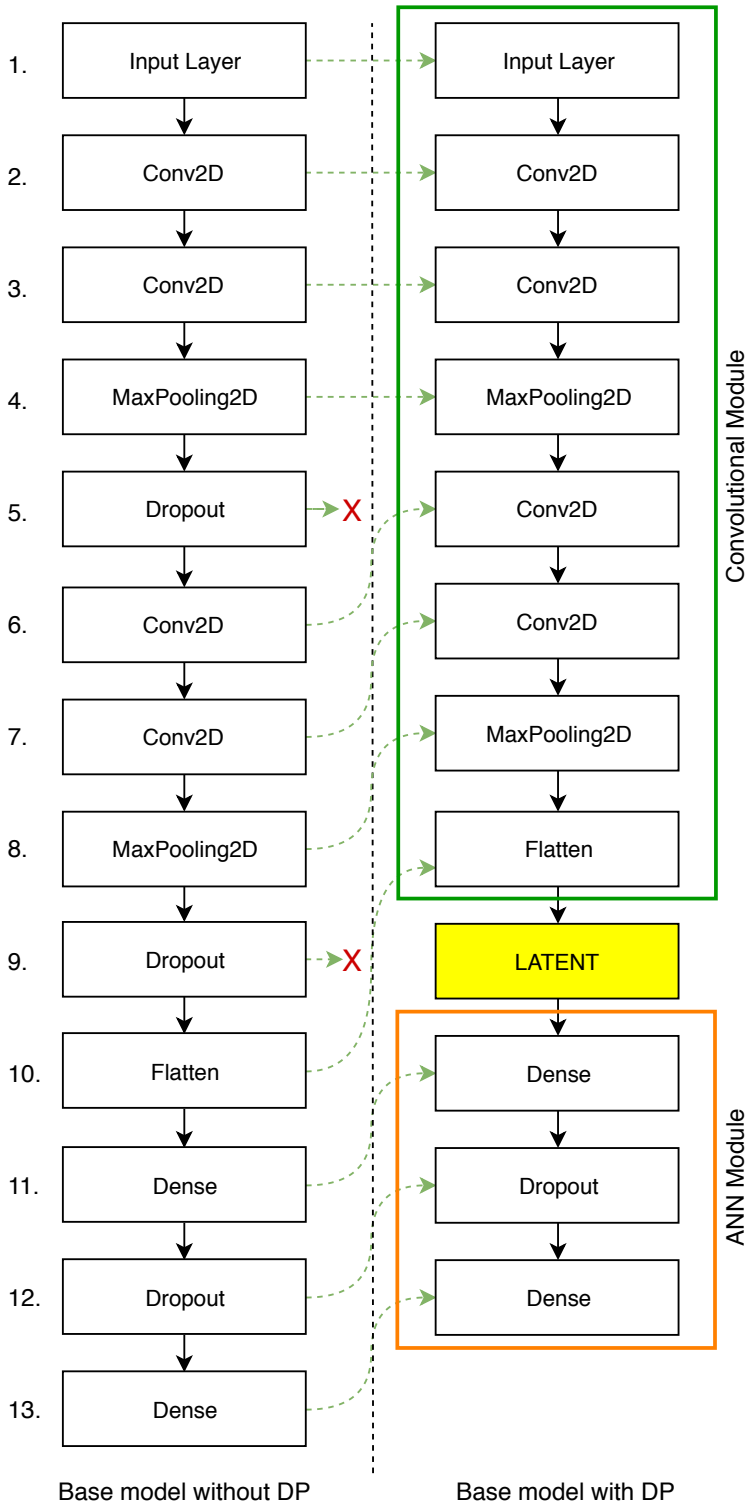

Figure 6: Architectural differences between the baseline models (NPCNN and DPCNN) for image recognition of the CIFAR-10 dataset

The values used for the hyperparameters during the HPT processes with each dataset (MNIST and CIFAR-10) are given in Table 1. We preferred higher number of neurons and epochs for CIFAR-10 due to its complexity compared to MNIST. During the hyperparameter tuning process, the probability of randomization (p) was set to 1, which corresponds to the nonprivate state of LATENT. When p=1, the binary feature vectors will not be randomized, and the model will result in greater accuracy than when p¡1. Due to the enlarged feature space compared to the conventional 1-d output of the flattening layer, the input features will have more representative properties. These properties will allow the proposed architecture of CNN to generate better accuracy than that of the original CNN architecture without LATENT. We generated the test cases using combinations of parameter values. We applied k-fold cross-validation (k=10) on each test case to derive a fair set of accuracy results. Since the search space is large, we divided the list of test cases into nine groups based on the combinations of the optimizers and the activation function. The best parameter values returned for each dataset are colored in red in Table 1. Next, we used the best parameters returned by the tuning process to produce differentially private models (for MNIST and CIFAR-10) with the best performance. We used the final DPCNN models of the hyperparameter tuning process to carry out further experiments and analyses.

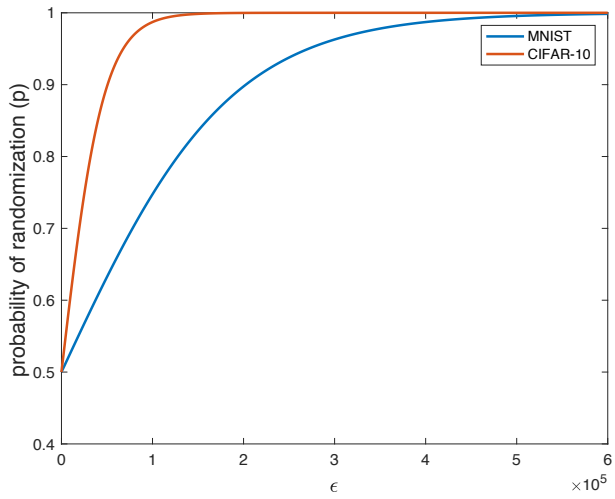

Figure 7: Change of p vs. ϵ for the two differentially private CNN models of the MINIST and CIFAR-10 datasets

4.1.3. Image augmentation to improve robustness of the DPCNN trained using CIFAR-10

Although we could improve the accuracy of the DPCNN for CIFAR-10 using hyperparameter tuning, the model was still not performing well and tended to overfit producing a training accuracy around 98-99% and a testing accuracy about 75-80% under the best-chosen hyperparameter values. To improve the model robustness, we applied image augmentation and generated 50,000 additional

augmented images using the 50,000 training input images. Each augmented image was generated by applying a random horizontal shift of a 0.1 fraction of the total width, a random vertical shift of a 0.1 fraction of the total height, a random rotation of 10 degrees, and a random horizontal flip, on the original input images. After introducing the new augmented images, the DPCNN model stopped overfitting and started generating a training accuracy of around 98% and a testing accuracy of about 95% consistently for repeated attempts (under a randomization probability of 1).

4.1.4. Selection of ϵ

As we discussed in Section 3.1, the probability of randomization can be given by Equation (6). When $l = 10$ for the DPCNN architecture defined for the MNIST dataset (depicted in Figure 5), $p = \frac{e^{\epsilon/92160}}{1+e^{\epsilon/92160}}$ as the sensitivity of the DP mechanism is 92160. For the DPCNN defined for the CIFAR-10 dataset (depicted in Figure 6), $p = \frac{e^{\epsilon/23040}}{1+e^{\epsilon/23040}}$, since the sensitivity is 23040. We can plot the change of p against ϵ for the two models as shown in Figure 7. As shown in the plots, the probability of randomization (p) lies around 0.5 for the acceptable values of ϵ (less than 10) as the sensitivity of the processes for both the models are large. Consequently, for the single digit positive ϵ values, p is persistent at 0.5. Hence, we use $\epsilon = 0.5$ to generate the results for all the experiments.

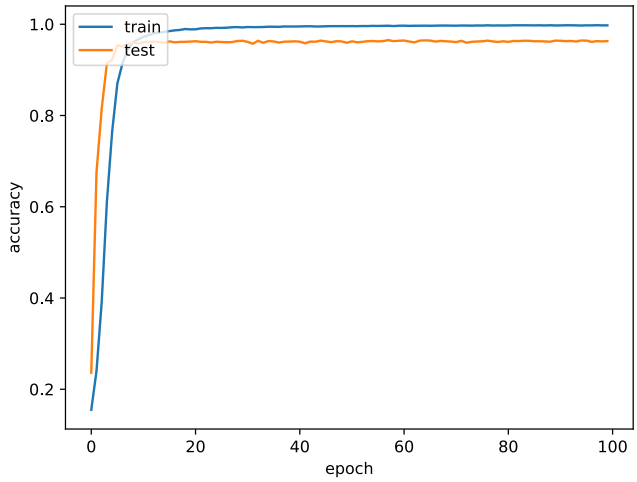

Figure 8: The change of the accuracy vs. the number of epochs during the training of the ANN module for the MNIST dataset (under $\epsilon = 0.5$ and the chosen hyper-parameters which are red-colored in Table 1 for MNIST)

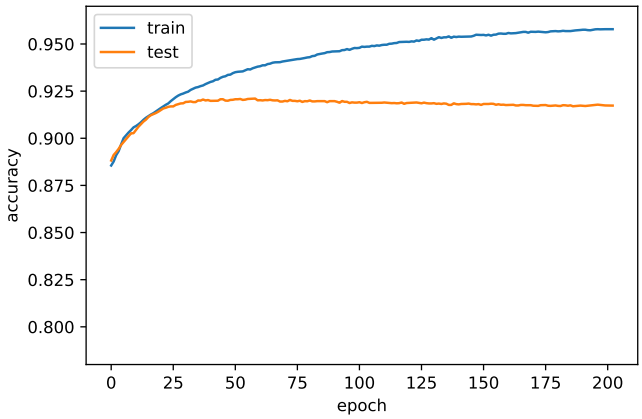

Figure 9: The change of accuracy vs. the number of epochs during the training of the ANN module for the CIFAR-10 dataset (under $\epsilon = 0.5$ and the chosen hyper-parameters which are red-colored in Table 1 for CIFAR-10)

Figure 8 shows the change of accuracy against the number of epochs during the training process of the ANN module on the MNIST dataset when $\epsilon = 0.5$. The ANN module converges at under 20 epochs to provide excellent training and testing accuracies. Clarity of the MNIST dataset and the availability of a large feature space generated by LATENT allow the model to produce excellent accuracies (Training: around 99% and Testing: around 96%) even at very low ϵ values such as 0.5 under the chosen hyper-parameters which are red-colored in Table 1.

Figure 9 shows the change of accuracy against the number of epochs during the training process of the ANN module of the CIFAR-10 dataset when $\epsilon = 0.5$. After applying image augmentation to 50,000 new images under the best-chosen hyper-parameters (red-colored in Table 1), the trained model returned around 96% training accuracy and about 91% testing accuracy after 200 epochs. The significant feature space generated by LATENT, and the large input space created by image augmentation allow the final model to produce the corresponding excellent accuracies with high robustness of the DP ANN model.

Figure 10 shows the change of accuracy against ϵ values. As the figure depicts, accuracy is almost constant although ϵ is changed. Recall that the probability of randomization is loosely affected by small values of ϵ due to the high sensitivity values as depicted in Figure 7. LATENT applies the highest possible randomization on each dataset under each case of ϵ depicted in Figure 10 and produces similar accuracy for smaller values of ϵ (< 10).

Table 2: Accuracy comparison of the results of LATENT against the existing methods

Dataset		**NPCNN**	**[SS15] [7]**	**[ACG+16] [1]**		**LATENT**	
		accuracy of the model without privacy	ϵ is large as it is reported per parameter	$\epsilon = 2$ $\delta = 10^{-5}$	$\epsilon = 0.5$ $\delta = 10^{-5}$	$\epsilon = 2$	$\epsilon = 0.5$
MNIST	Training	99.25%	N/A	∼95%	∼89%	99.23%	99.42%
	Testing	98.16%	98%	95%	90%	96.34%	96.26%
CIFAR-10	Training	73.32%	N/A	∼68%	N/A	95.62%	95.77%
	Testing	78.75%	N/A	67%	N/A	91.73%	91.47%

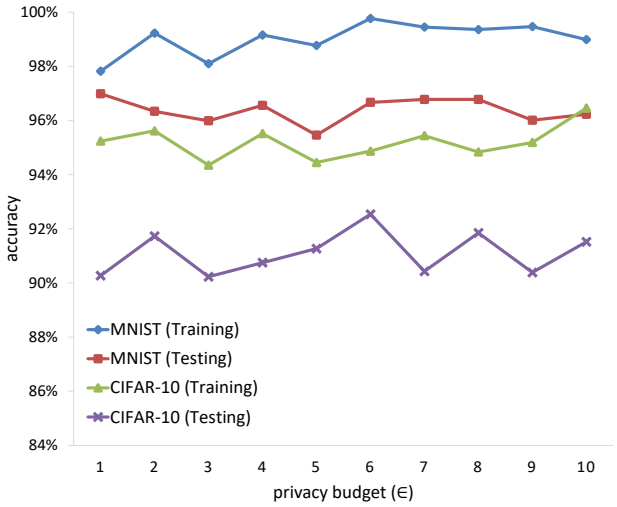

Figure 10: Change of accuracy of LATENT against ϵ

We compare our results with two other existing differentially private mechanisms for deep learning as shown in Table 2. Although [SS15] [7] provides good accuracy; ϵ is presented per parameter of the model. It can accumulate a large, unacceptable ϵ value at the end of model generation as there can be more than 1000 model parameters. For $\epsilon = 2$ and $\delta = 10^{-5}$ the [ACG+16] [1] method provides good accuracy, yet the additive bound of δ can become unreliable when the method is used for much larger datasets. [ACG+16] has failed to generate acceptable accuracy for the CIFAR-10 dataset under an extreme case like $\epsilon = 0.5$. Our method provides comparably much better accuracy under an extreme case such as $\epsilon = 0.5$. Also, the unavailability of additive bound ensures that our method has low privacy leak when substantially large input datasets are presented to the method. Both [SS15] and [ACG+16] are based on global differential privacy. Therefore, the availability of a trusted party is unavoidable. For a real-world scenario, often we don't have any trusted party. LATENT can be a much better solution in such cases, as it works in both untrusted curator and trusted curator scenarios. Table 3 sums up the advantages of LATENT over the existing GDP methods for deep learning.

Table 3: Advantages of LATENT over the existing GDP methods for deep learning

GDP methods	LATENT
Always needs a trusted curator.	LATENT can be used for both trusted and untrusted settings.
For machine learning with cloud computing, original data needs to be uploaded to the server considering the server is trustable. However, the servers cannot always be trusted.	LATENT randomizes data before uploading them to the server in case the server is not trustable.
A higher privacy loss (a larger privacy budget - ϵ) needs to be allocated to obtain a better utility.	LATENT provides excellent utility in terms of classification accuracy (more than 90%) even under an extreme level of randomization ($\epsilon = 0.5$).
GDP runs either in client side or a server side. The distrust of the server might prevent the algorithm being run on a server. However, deep learning algorithms tend to be complex and can be complex for a general purpose personal computer, and privacy preservation techniques often add more complexity. This feature reduces the practicality of GDP algorithms for deep learning.	As LATENT is an LDP algorithm, it doesn't have an obligation to have a trusted curator. As the proposed architecture is already a distributed version which utilizes the computational power of data owners and the servers, LATENT is more practical compared to GDP approaches.

4.2. Computational burden on resource-constrained users

In the proposed modular decomposition of the CNN architecture, the convolutional module and the LATENT module run on the data owner's machine. We need to make sure that the convolutional module and LATENT operations do not impose a substantial computational burden on the resource-constrained data owners. In order to check this, we measured the time consumption for the perturbation of a single record of MNIST and CIFAR-10 datasets separately on a MacBook Pro (macOS Mojave, 13-inch, 2017) computer with Intel Core i5 CPU (2.3 GHz), 8 GB RAM and 1536MB GPU (Intel Iris Plus Graphics). It took an average time of 0.1655 seconds to perturb a single record of MNIST dataset while consuming 0.0374 seconds to perturb a single record of CIFAR-10 dataset. This indicates that a a general purpose computer with moderate specification will suffice for generating the randomized data.

5. Related Work

A main challenge in privacy-preserving data mining (PPDM) is countering the capabilities of skilled adversaries [26, 27, 28, 29]. Data modification (data perturbation) [30, 31] and encryption [32, 33] are two main approaches to PPDM. Methods based on encryption provide good security and accuracy. However, cryptographic methods often suffer from high computational complexity which make them

unsuitable for large-scale data mining [34]. Compared to encryption, perturbation utilizes lower computational complexity, which makes it effective for big data mining [35]. Examples for perturbation techniques include noise addition [36], geometric transformation [31], randomization [12], condensation [37], hybrid perturbation (uses several perturbation techniques together) [38].

Data perturbation may allow some privacy leak since the data/results are released in their original format [35]. Hence, a privacy model should identify the limits of private information protection/disclosure mechanism [39]. Earlier privacy models include $k-anonymity$ [40], $l-diversity$ [41], $(\alpha, k)-anonymity$ [42], $t-closeness$ [43]. It has been shown that these models are vulnerable to different attacks such as minimality attack [44], composition based attacks [45] and foreground knowledge [46] attacks. Differential privacy (DP) is trusted to provide a better level of privacy guarantee compared to previous privacy models [47, 48, 49].

Laplace mechanism, Gaussian mechanism [19], geometric mechanism, randomized response [17], and staircase mechanisms [11] are a few of the fundamental mechanisms used to achieve differential privacy. There are many practical examples where these fundamental mechanisms have been used to build differentially private algorithm/methods. Differential Privacy for SQL Queries [50], LDPMiner [17], PINQ [51], RAPPOR [14], Succinct histogram [52] and Deep Learning with Differential Privacy [1] are a few examples of such practical applications.

Literature shows a few attempts to address the issue of privacy leaks in deep learning algorithms by imposing private training [1, 7, 53, 54, 55]. Shokri, R. et al. [7] developed a distributed multiparty learning mechanism (referred to as [SS15] in Table 2) for a neural network without sharing input datasets. They parallelized the learning process which is based on the stochastic gradient descent optimization algorithm. The main advantage of their method is the ability of participants to preserve the privacy of their respective data while still benefiting from other participants' models which are shown to achieve high learning accuracy. They compute the privacy loss per parameter of the model. This can entail a substantial privacy loss as there are many model parameters, often there can be thousands of such model parameters. Abadi, M. et al. [1] introduced an efficient differentially private mechanism (referred to as [ACG+16] in Table 2) based on global differential privacy. Their model is capable of achieving high efficiency and performance under a modest privacy budget. Their algorithm is based on a differentially private version of stochastic gradient descent which runs on the TensorFlow software library for machine learning. Further, they introduced a tool to track privacy loss, the moments accountant, which allows tight automated analysis of privacy loss. But the additive

bound δ of their (ϵ,δ)-differential privacy mechanism may incur an unreliable level of privacy leak when the method is used for much larger datasets. Another shortcoming of the two methods [SS15] and [ACG+16] is the need for a trusted third party. Since both methods are based on global differential privacy, the necessity of having a trusted third party cannot be avoided. This can be considered as a significant issue in applying these methods to real-world scenarios, where trusted curators are not always available.

LATENT is designed to be aligned with machine learning as a service scenario which has become popular due to the capabilities offered by large Internet-based companies, such as Google and Amazon [3]. For example Google's cloud-based machine learning engine provides the ability to build the models with multiple ML frameworks such as scikit-learn [56], XGBoost [57], Keras [24], and TensorFlow [3]. LATENT uses similar technologies for its implementation, replicating the technical settings of the environment offered by Google's cloud ML platform and other related services. Consequently, the model evaluation would adhere to the same sequence of stages enabled by the Google cloud ML engine, allowing to generalize the results upon such online DL model services with black-box access to them. LATENT provides excellent accuracy under extreme cases of privacy, maintaining an outstanding balance between privacy and utility.

6. Conclusion

We proposed a new local differentially private mechanism to train a deep neural network with high privacy and high accuracy. Our model exhibits remarkably excellent accuracy even under extreme cases of privacy (e.g. $\epsilon = 0.5$) compared to the existing differentially private approaches. We achieve 96% testing accuracy and 91% testing accuracy for the MNIST dataset and CIFAR-10 dataset respectively with an outstanding level of privacy (0.5-differential privacy). Due to the large feature space created by LATENT during the randomization process, it generates better accuracy for CIFAR-10 dataset even than the baseline CNN model without any privacy. Existing differentially private mechanisms are implemented using global differential privacy, and so they need a trusted curator. The untrusted curator setting guarantees that our approach provides a higher level of privacy while leaving a low level of computational burden to the data owners. Moving the convolutional module to the data owners produces additional privacy even without the application of randomization as the convolutional module output is a 1-d dimension-reduced output. The distribution of the CNN structure between data owners and servers also increases the flexibility of data processing in the big data context. When

a large number of data owners communicate with a single server, the server has to be concerned only about generating the differentially private ANN model. The ability to use our method in the untrusted curator setting allows the private sharing of sensitive data and limits the privacy leak in distributed machine learning scenarios. Since the proposed method is based on LDP, we do not make any architectural modifications to the fully connected artificial neural network component (which we call the ANN module) of a convolutional network. Therefore, the input parameter selection (e.g. ϵ, number of input bits) of the differentially private component (LATENT) is independent of the tuning processes (e.g. regularization, image augmentation, and hyperparameter tuning) of the ANN module in the CNN architecture. This allows easy training and tuning of the ANN module with a higher level of accuracy and an extreme level of privacy, resulting in an outstanding balance between privacy and utility.

Our approach opens up many future research directions. Investigating the possibility of reducing the data sensitivity would be a good research avenue. Low sensitivity would allow the selection of an appropriate ϵ value tailored to the domain requirements. We would also like to test our method on other deep learning architectures such as recurrent networks with LSTM (Long Short-Term Memory) and test it for other large datasets to find its performance and generalizability.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, L. Zhang, Deep learning with differential privacy, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 308–318.

[2] R. Shokri, M. Stronati, C. Song, V. Shmatikov, Membership inference attacks against machine learning models, in: Security and Privacy (SP), 2017 IEEE Symposium on, IEEE, 2017, pp. 3–18.

[3] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al., Tensorflow: a system for large-scale machine learning., in: OSDI, Vol. 16, 2016, pp. 265–283.

[4] Y. Low, D. Bickson, J. Gonzalez, C. Guestrin, A. Kyrola, J. M. Hellerstein, Distributed graphlab: a framework for machine learning and data mining in the cloud, Proceedings of the VLDB Endowment 5 (8) (2012) 716–727.

[5] C. Song, T. Ristenpart, V. Shmatikov, Machine learning models that remember too much, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2017, pp. 587–601.

[6] M. Fredrikson, S. Jha, T. Ristenpart, Model inversion attacks that exploit confidence information and basic countermeasures, in: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, ACM, 2015, pp. 1322–1333.

[7] R. Shokri, V. Shmatikov, Privacy-preserving deep learning, in: Proceedings of the 22nd ACM SIGSAC conference on computer and communications security, ACM, 2015, pp. 1310–1321.

[8] G. Cormode, S. Jha, T. Kulkarni, N. Li, D. Srivastava, T. Wang, Privacy at scale: Local differential privacy in practice, in: Proceedings of the 2018 International Conference on Management of Data, ACM, 2018, pp. 1655–1658.

[9] C. Dwork, A. Roth, et al., The algorithmic foundations of differential privacy, Foundations and Trends® in Theoretical Computer Science 9 (3–4) (2014) 211–407.

[10] X. Xiao, Y. Tao, Output perturbation with query relaxation, Proceedings of the VLDB Endowment 1 (1) (2008) 857–869.

[11] P. Kairouz, S. Oh, P. Viswanath, Extremal mechanisms for local differential privacy, in: Advances in neural information processing systems, 2014, pp. 2879–2887.

[12] J. A. Fox, Randomized response and related methods: Surveying Sensitive Data, Vol. 58, SAGE Publications, 2015.

[13] T.-H. H. Chan, M. Li, E. Shi, W. Xu, Differentially private continual monitoring of heavy hitters from distributed streams, in: International Symposium on Privacy Enhancing Technologies Symposium, Springer, 2012, pp. 140–159.

[14] Ú. Erlingsson, V. Pihur, A. Korolova, Rappor: Randomized aggregatable privacy-preserving ordinal response, in: Proceedings of the 2014 ACM SIGSAC conference on computer and communications security, ACM, 2014, pp. 1054–1067.

[15] S. L. Warner, Randomized response: A survey technique for eliminating evasive answer bias, Journal of the American Statistical Association 60 (309) (1965) 63–69.

[16] Y. Wang, X. Wu, D. Hu, Using randomized response for differential privacy preserving data collection., in: EDBT/ICDT Workshops, Vol. 1558, 2016.

[17] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, K. Ren, Heavy hitter estimation over set-valued data with local differential privacy, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2016, pp. 192–203.

[18] M. Bun, T. Steinke, Concentrated differential privacy: Simplifications, extensions, and lower bounds, in: Theory of Cryptography Conference, Springer, 2016, pp. 635–658.

[19] T. Chanyaswad, A. Dytso, H. V. Poor, P. Mittal, Mvg mechanism: Differential privacy under matrix-valued query, arXiv preprint arXiv:1801.00823.

[20] J. Schmidhuber, Deep learning in neural networks: An overview, Neural networks 61 (2015) 85–117.

[21] Y. Kim, Convolutional neural networks for sentence classification, arXiv preprint arXiv:1408.5882.

[22] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, 2012, pp. 1097–1105.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, The Journal of Machine Learning Research 15 (1) (2014) 1929–1958.

[24] F. Chollet, et al., Keras: Deep learning library for theano and tensorflow, URL: https://keras.io/k 7 (8).

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

[26] M. Xue, P. Papadimitriou, C. Raïssi, P. Kalnis, H. K. Pung, Distributed privacy preserving data collection, in: International Conference on Database Systems for Advanced Applications, Springer, 2011, pp. 93–107.

[27] M. Backes, P. Berrang, O. Goga, K. P. Gummadi, P. Manoharan, On profile linkability despite anonymity in social media systems, in: Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society, ACM, 2016, pp. 25–35.

[28] K. Yang, Q. Han, H. Li, K. Zheng, Z. Su, X. Shen, An efficient and fine-grained big data access control scheme with privacy-preserving policy, IEEE Internet of Things Journal 4 (2) (2017) 563–571.

[29] D. Vatsalan, Z. Sehili, P. Christen, E. Rahm, Privacy-preserving record linkage for big data: Current approaches and research challenges, in: Handbook of Big Data Technologies, Springer, 2017, pp. 851–895.

[30] K. Chen, L. Liu, A random rotation perturbation approach to privacy preserving data classification.
URL https://corescholar.libraries.wright.edu/knoesis/916/

[31] K. Chen, L. Liu, Geometric data perturbation for privacy preserving outsourced data mining, Knowledge and Information Systems 29 (3) (2011) 657–695.

[32] J. Li, D. Lin, A. C. Squicciarini, J. Li, C. Jia, Towards privacy-preserving storage and retrieval in multiple clouds, IEEE Transactions on Cloud Computing 5 (3) (2017) 499–509. doi:10.1109/TCC.2015.2485214.

[33] F. Kerschbaum, M. Härterich, Searchable encryption to reduce encryption degradation in adjustably encrypted databases, in: IFIP Annual Conference on Data and Applications Security and Privacy, Springer, 2017, pp. 325–336.

[34] K. Gai, M. Qiu, H. Zhao, J. Xiong, Privacy-aware adaptive data encryption strategy of big data in cloud computing, in: Cyber Security and Cloud Computing (CSCloud), 2016 IEEE 3rd International Conference on, IEEE, 2016, pp. 273–278.

[35] H. Xu, S. Guo, K. Chen, Building confidential and efficient query services in the cloud with rasp data perturbation, IEEE transactions on knowledge and data engineering 26 (2) (2014) 322–335.

[36] K. Muralidhar, R. Parsa, R. Sarathy, A general additive data perturbation method for database security, management science 45 (10) (1999) 1399–1415.

[37] C. C. Aggarwal, P. S. Yu, A condensation approach to privacy preserving data mining, in: EDBT, Vol. 4, Springer, 2004, pp. 183–199.

[38] M. A. P. Chamikara, P. Bertok, D. Liu, S. Camtepe, I. Khalil, Efficient data perturbation for privacy preserving and accurate data stream mining, Pervasive and Mobile Computing`doi:10.1016/j.pmcj.2018.05.003`.

[39] A. Machanavajjhala, D. Kifer, Designing statistical privacy for your data, Communications of the ACM 58 (3) (2015) 58–67.

[40] B. Niu, Q. Li, X. Zhu, G. Cao, H. Li, Achieving k-anonymity in privacy-aware location-based services, in: INFOCOM, 2014 Proceedings IEEE, IEEE, 2014, pp. 754–762.

[41] A. Machanavajjhala, J. Gehrke, D. Kifer, M. Venkitasubramaniam, l-diversity: Privacy beyond k-anonymity, in: Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on, IEEE, 2006, pp. 24–24.

[42] R. C.-W. Wong, J. Li, A. W.-C. Fu, K. Wang, (α, k)-anonymity: an enhanced k-anonymity model for privacy preserving data publishing, in: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2006, pp. 754–759.

[43] N. Li, T. Li, S. Venkatasubramanian, t-closeness: Privacy beyond k-anonymity and l-diversity, in: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on, IEEE, 2007, pp. 106–115.

[44] L. Zhang, S. Jajodia, A. Brodsky, Information disclosure under realistic assumptions: Privacy versus optimality, in: Proceedings of the 14th ACM conference on Computer and communications security, ACM, 2007, pp. 573–583.

[45] S. R. Ganta, S. P. Kasiviswanathan, A. Smith, Composition attacks and auxiliary information in data privacy, in: Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2008, pp. 265–273.

[46] R. C.-W. Wong, A. W.-C. Fu, K. Wang, P. S. Yu, J. Pei, Can the utility of anonymized data be used for privacy breaches?, ACM Transactions on Knowledge Discovery from Data (TKDD) 5 (3) (2011) 16.

[47] C. Dwork, The differential privacy frontier, in: Theory of Cryptography Conference, Springer, 2009, pp. 496–502.

[48] N. Mohammed, R. Chen, B. Fung, P. S. Yu, Differentially private data release for data mining, in: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2011, pp. 493–501.

[49] A. Friedman, A. Schuster, Data mining with differential privacy, in: Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2010, pp. 493–502.

[50] N. Johnson, J. P. Near, D. Song, Towards practical differential privacy for sql queries, Proceedings of the VLDB Endowment 11 (5) (2018) 526–539.

[51] F. D. McSherry, Privacy integrated queries: an extensible platform for privacy-preserving data analysis, in: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data, ACM, 2009, pp. 19–30.

[52] R. Bassily, A. Smith, Local, private, efficient protocols for succinct histograms, in: Proceedings of the forty-seventh annual ACM symposium on Theory of computing, ACM, 2015, pp. 127–135.

[53] P. Li, J. Li, Z. Huang, T. Li, C.-Z. Gao, S.-M. Yiu, K. Chen, Multi-key privacy-preserving deep learning in cloud computing, Future Generation Computer Systems 74 (2017) 76–85.

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

[55] S. A. Osia, A. S. Shamsabadi, A. Taheri, K. Katevas, S. Sajadmanesh, H. R. Rabiee, N. D. Lane, H. Haddadi, A hybrid deep learning architecture for privacy-preserving mobile analytics, arXiv preprint arXiv:1703.02952.

[56] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al., Scikit-learn: Machine learning in python, Journal of machine learning research 12 (Oct) (2011) 2825–2830.

[57] T. Chen, C. Guestrin, Xgboost: A scalable tree boosting system, in: Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining, ACM, 2016, pp. 785–794.