Bachelor of Science in Electrical and Electronic Engineering

EEE 400 (January 2024): Thesis

Implementation and Performance Analysis of DPMix Algorithm – a Random Mixing Technique for Synthesizing Differentially Private Datasets

Submitted by

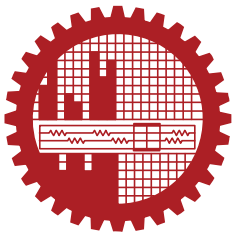
Sabir Mahmud

201906032

Supervised by

Dr. Hafiz Imtiaz

Professor



Department of Electrical and Electronic Engineering

Bangladesh University of Engineering and Technology

Dhaka, Bangladesh

March 2025

CANDIDATES' DECLARATION

This is to certify that the work presented in this thesis, titled, "Implementation and Performance Analysis of DPMix Algorithm – a Random Mixing Technique for Synthesizing Differentially Private Datasets", is the outcome of the investigation and research carried out by us under the supervision of Dr. Hafiz Imtiaz.

It is also declared that neither this thesis nor any part thereof has been submitted anywhere else for the award of any degree, diploma or other qualifications.

Sabir Mahmud
201906032

CERTIFICATION

This thesis titled, **"Implementation and Performance Analysis of DPMix Algorithm – a Random Mixing Technique for Synthesizing Differentially Private Datasets"**, submitted by the group as mentioned below has been accepted as satisfactory in partial fulfillment of the requirements for EEE 400: Project/Thesis course, and as the requirements for the degree B.Sc. in Electrical and Electronic Engineering in March 2025.

Group Members:

Sabir Mahmud

Supervisor:

Dr. Hafiz Imtiaz
Professor
Department of Electrical and Electronic Engineering
Bangladesh University of Engineering and Technology

ACKNOWLEDGEMENT

I have been learning and exploring the field of Differential Privacy under the supervision of Professor Dr. Hafiz Imtiaz for over a year. He has been guiding me with necessary information, research papers, and resources for learning deep learning algorithms, as well as providing proper directions to develop this thesis. His guidance has been greatly helpful to me for my research and in successful completion of this thesis. I am deeply thankful to him for the time and knowledge he has provided me throughout my research work.

I would also like to express my gratitude to all of our teachers, classmates and seniors for the valuable knowledge and encouragement they provided me and made my academic journey easy and beautiful.

Sabir Mahmud

Dhaka
March 2025

Contents

List of Figures

List of Tables

ABSTRACT

In modern world, proper implementation of network connectivity and other digital systems require personal data for various purposes. Many government and non-government organizations need our private and day to day data for running their activities. For example, academic, medical and military sectors might need some data from the population which we might consider personal and we would feel insecure in giving up such private data for their research work. To solve such problems, a special security system was developed called 'Differential Privacy'. Differential Privacy is a privacy preserving technique that conceals individual data points in a dataset by adding controlled random noise. Through the technique we can achieve a synthetic dataset that has almost similar attributes and distribution of the original dataset, but is not exactly identical to the original one. This new synthetic dataset can be later used in various learning and training techniques and can be used to determine any statistical pattern of certain behavior within the population based on their personal attributes. In this thesis we explore a special Random Mixing technique called DPMix algorithm, that can achieve significant privacy while ensuring that the synthetic dataset gives nearly accurate outputs as the original one.

Chapter 1

Introduction

1.1 Privacy in Digital Systems

Today businesses are utilizing more and more of our information to make their products and services better. It's actually necessary for them, because it's better to quantify what users enjoy than to make assumptions and create products that will be enjoyed by a few random customers. But this is also extremely risky, it destroys our privacy because the aggregated data can be really sensitive, causing harm if it would leak.

Some would say that the solution is simple—just take the data and strip out the identities of the users. It isn't quite as easy as it seems. In the first place, this anonymizing is typically being performed on the servers of those companies collecting the data, so people have to trust the firms that process their identifiable records into oblivion. In the second place, it is not at all clear how anonymous is the anonymized data in reality.

In 2006 Netflix sponsored a competition called the Netflix prize. The contestants were asked to create an algorithm that would predict how many stars a person would give to a movie. To help with this task Netflix provided a dataset containing over 100 million ratings from over 480 thousand users on over 17 thousand movies. Netflix of course anonymized this data set by erasing the usernames and substituting some of the ratings with fake and random ratings. That, initially had sounded very anonymous, but in fact it was not. In 2008, two computer scientists at the University of Texas published a paper indicating that they'd been able to identify individuals based on this data set by matching it against data from IMDB [1]. This is called linkage attacks and it happens when pieces of seemingly anonymous information can be combined to reveal real identities [2].

Another creepier example would be that of the governor of Massachusetts. Its Group Insurance Commission in the mid-1990s made the decision to release the hospital visits of the state employees. They made the data anonymous by deleting names, addresses and other identifying

fields. However, computer scientist Latanya Sweeney decided to prove how easy it was to reverse that process. She combined the released health records with voter registration records and then easily consolidated the list. She found out that, there was only one person in the medical records who lived in the same zip code, was the same gender, and had the same date of birth as the governor, thus revealing his medical records. Subsequently, she pointed out that 87% of all Americans could be singled out using only three pieces of information: zip code, birthday, and gender [3]. It thus becomes glaringly apparent that this method does not safeguard our privacy. A recently developed algorithm named 'Differential Privacy' can solve this problem and can neutralize these types of attacks on privacy.

1.2 Differential Privacy

Various private and governmental institutions, like hospitals and academic centers routinely collect large amount of personal data from the individuals who are their clients, customers and who access their services. These data become necessary to further predict the pattern of any certain topic of interest, like diagnosis of diseases or academic outcomes etc. But much of this information is private or sensitive. Companies tend to utilize the personal data of the clients more and more, while the users want to protect their privacy. So, a big challenge comes for the organizations to utilize those data efficiently while preserving the privacy of these data and their owners.

In recent years, various privacy preserving techniques have been developed to utilize the available data while protecting the privacy of the related individuals. One of these techniques received much attention because of its strong privacy protection ability and efficiency over digitized dataset. Giant companies like Apple and Google are already using this technique and putting continuous effort on its development and increment of its effectiveness.

Differential privacy promises that the outcome of a survey will stay the same whether you participate or not in it. Therefore, there is no reason for anyone to participate in it. Roughly, an algorithm is differentially private if an observer - after seeing its output can't tell whether a particular individual's data was used in the computation or not. Differential privacy is often explained in terms of discovering individuals whose data might be in a database. It does not specifically talk about identification and reidentification attacks, but differentially private algorithms strongly defend against them [4].

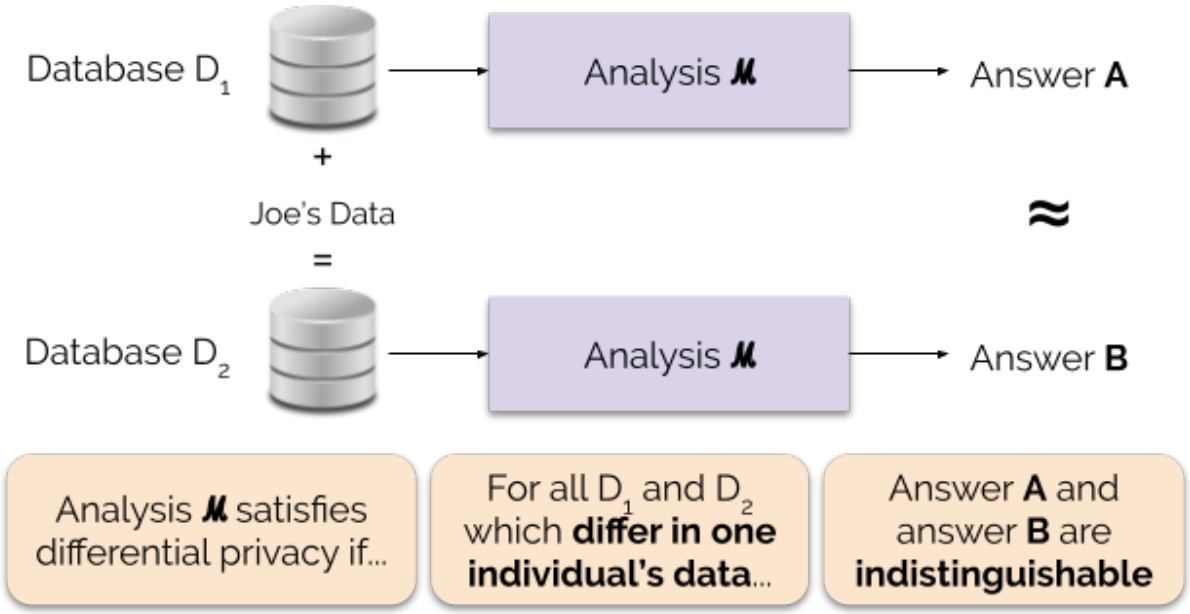


Figure 1.1: An informal definition of differential privacy

This thesis work is conducted with a focus on the topic of differential privacy.

1.3 ε-differential privacy

The research article by Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith published in 2006 [5] introduced the concept of ε-differential privacy, a rigorous definition for the privacy loss of any data release from a statistical database. Here, by statistical database, we refer to a set of data which are collected under the guarantee of confidentiality with a view to producing statistics, that, by their creation, do not infringe the privacy of those individuals who submitted the data [6]. The ε-differential privacy notion mandates that a modification of a single entry in a database causes only a slight change in the distribution of the outputs of measurements [5].

In differential privacy, all parties are given roughly the same privacy that one would receive when his or her data is removed [7]. That is, the database should not be greatly affected in regard to its statistical functions due to the removal, addition, or alteration of any individual in the data [7].

Let ε be a positive real number and $\mathcal{A}$ be a randomized algorithm that takes a dataset as input (representing the actions of the trusted party holding the data). Let *im* $\mathcal{A}$ denote the image of $\mathcal{A}$.

The algorithm $\mathcal{A}$ is said to provide (ε, δ)-differential privacy if, for all datasets D_1 and D_2 that differ on a single element (i.e., the data of one person), and all subsets $\mathcal{S}$ of *im* $\mathcal{A}$:

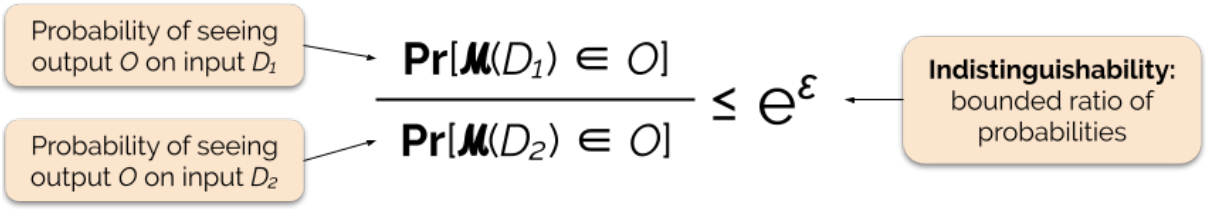


Figure 1.2: A formal definition of ε-differential privacy.
Here, in the above figure, D_1 is a dataset without the private data, and D_2 is a dataset containing the private data. This is "pure ε-differential privacy", meaning $\delta = 0$.

$$\Pr[\mathcal{A}(D_1) \in \mathcal{S}] \leq e^{\varepsilon} \Pr[\mathcal{A}(D_2) \in \mathcal{S}] + \delta$$

1.4 DPMix Algorithm – a Random Mixing Technique

In 2019 a new privacy preserving algorithm named 'DPMix' was proposed by Kangwook Lee, Hoon Kim, Kyungmin Lee, and Changho Suh in their research paper titled with *'Synthesizing Differentially Private Datasets using Random Mixing'* [1]. In this paper they introduced a deep learning algorithm, where a new synthetic dataset is formed by mixing ℓ randomly chosen data points from the original dataset and then perturbing them with an additive noise. They also showed that there exists a sweet spot on ℓ that maximizes the prediction accuracy given a required privacy level, and vice versa.

1.5 Outline

This thesis work focuses on the implementation and analysis of a special privacy preserving technique named 'DPMix algorithm' introduced by Lee et al. in 2019.

The thesis book is organized as followed: Chapter 1 contains introduction of Differential privacy and its usefulness. Chapter 2 contains literature review of the related works. Chapter 3 contains the methodology and necessary formulas related to the implementation of DPMix algorithm. Chapter 4 contains the analyzation of the outputs I got from my coding and effects of various parameters on the process. Finally, chapter 5 concludes the results I got by the implementation of the deep learning model and explore its usability for future works and further development of this algorithm.

Chapter 2

Literature Review

The issue of data collection privacy has been a subject of significant interest, especially with the introduction of new privacy laws like the **EU General Data Protection Regulation (GDPR)**. In response to such concerns, researchers have been exploring **Privacy-Preserving Machine Learning (PPML)** methods that possess the potential to train machine learning models without directly accessing the privacy-sensitive data.

One such prominent PPML technique is **Federated Learning (McMahan et al., 2017)** [9]. It enables the training of neural networks without the exchange of raw data by sharing gradients (mathematical updates) only. Yet, it was demonstrated by researchers **(Zhu & Han, 2020)** that the adversary can recover the original data by deducing it from the gradients [10], and hence federated learning is not entirely safe.

Similarly, **Split Learning (Vepakomma et al., 2018)** divides a machine learning model into two halves, with one being kept by the data owner and the other by the central server [11]. It must be privacy-respecting, the reasoning goes, to send extracted features (as opposed to raw data) across. But it was shown by **(He et al., 2019)** that it can be possible for the adversaries to reconstruct shared feature space to retrieve the initial data [12].

Both techniques involve the release of altered information, but because the transformations can typically be reversed, they offer very little privacy guarantees. Shortcomings of these techniques can be overcome by applying the techniques of Differential Privacy.

Differential Privacy (DP)

Differential Privacy (Dwork et al., 2014) is an established model for privacy [13] used across both academia and industry. There are several major companies that have used DP, including:

- **Google** (RAPPOR, Erlingsson et al., 2014) [14]

- **LinkedIn** (PriPeARL, Kenthapadi & Tran, 2018) [15]
- **Apple** (Apple, 2017) [16]
- **Microsoft** (Ding et al., 2017) [17]

DP ensures that the presence or exclusion of any specific record in a set does not have any major influence on the result produced by a function. Which makes it nearly impossible for the attacker to identify whether the information of any specific user was included or not. In order to utilize DP for machine learning, it is proposed to build a differentially private dataset. Among the most notable approaches of DP are:

- **PrivBayes (Zhang et al., 2017b)** makes use of Bayesian networks to generate synthetic records with DP guarantees [18].
- **DPPro (Xu et al., 2017)** reduces dimensionality of data for privacy enhancement [19].
- **DP-GANs (Xie et al., 2018; Zhang et al., 2018)** Uses Generative Adversarial Networks (GANs) for generating differentially private datasets [20, 21].
- **P3GM (Takagi et al., 2021)** is a privacy-ensuring generative model for high dimensional data [22].

However, most of these methods break down with high-dimensional datasets that contain many features. For example:

DPPro and **PrivBayes** are not scalable to high-dimensional feature sets.

Now, we will learn about some works that are related to this thesis work. We will be introduced with some of the basics of (ε,δ)-Differential Privacy, differentially private data release algorithms, DPSGD, mixup and their applications in privacy protection. We will also be introduced with two threats (i.e. attacking algorithms) against privacy protecting algorithms.

2.1 (ε, δ)-Differential Privacy

(ε, δ)-DP offers privacy by adding noise to calculations such that the adversary can't identify specific points. (ε, δ)-differential privacy **(Dwork et al., 2006)** [23] is one of the most widely accepted approach of privacy. A function is (ε, δ)-DP if the probability of any specific outcome does not alter much when the individual record is altered. This can be expressed by the following formula:

$$\Pr[\mathbf{A}(D_1) \in S] \leq e^{\varepsilon} \Pr[\mathbf{A}(D_2) \in S] + \delta$$

ε and δ are privacy parameters.

In order to achieve (ε, δ)-DP, we add Gaussian noise to data operations. The amount of noise we need is calculated by sensitivity, which quantifies how much the output of a function can change based on an individual data point.

There exist two major benefits of (ε, δ)-DP:

1. **Robustness to post-processing:** Any transformation of the output of any differentially private function is still differentially private.
2. **Privacy loss accounting:** DP provides a mathematically sound method for accounting aggregated privacy loss for many different calculations.

However, standard (ε, δ)-DP is not always universal for every use scenario. Stronger privacy guarantees are offered by some relaxed variants like **Renyi-DP (Mironov, 2017)** [24] and **Gaussian Differential Privacy (GDP, Dong et al., 2021)** [25].

2.2 Differentially Private Data Release

Many different DP-based methods for releasing individual information and its utility to machine learning have been proposed by researchers.

- **PrivBayes (Zhang et al., 2017)** - Builds noise-injected probabilistic model to generate synthetic datasets [18].
- **DPPro (Xu et al., 2017)** reduces the feature count to limit privacy attacks [19].
- **NIST-MST (McKenna et al., 2021)** is an effective method that introduces noise into the data and subsequently recreates realistic distributions [26].
- **Generative Models** such as GANs and VAEs have also been adapted to produce DP-protected synthesized data but are plagued by mode collapse (the synthesized data lacks diversity).

DPMix (Lee et al., 2019) is new in that it averages small subsets of the data with added noise [8]. It does offer an appropriate privacy-utility trade-off, but the appropriate subset size needs to be selected.

2.3 Differentially Private Stochastic Gradient Descent (DPSGD)

DPSGD (Abadi et al., 2016) [27] is a privacy-assuring training method that introduces noise into the gradients. It limits the effect of any single data point on the model. However, DPSGD does reduce the model's accuracy, when contrasted with standard training methods.

Some researchers have attempted to improve the DPSGD technique through **Feature extraction (Tramer & Boneh, 2021)**, which uses pre-trained models instead of training anew for privacy-utility tradeoff improvement [28].

2.4 Mix-Up for Privacy Protection

Mixup (H. Zhang et al., 2017) is an image blending method during training [29]. It reduces the problem of overfitting along with the easiness with which the adversary can deduce information that is private.

Mix-up extensions for privacy include:

- **AdaMixUp (Chidambaram et al., 2021):** Learns the optimal blending strategies [30].
- **DPMix (Lee et al., 2019):** Utilizes mixed samples so that original data cannot be recovered by attackers [8].
- **InstaHide (Huang et al., 2020):** It encrypts the image first [31]. However, this method was later compromised by **Carlini et al. (2020)** that shows that this method is somewhat insecure [32].

2.5 Attacks on Privacy-Preserving Models

Even with privacy protection, the attacker can try to gain personal information through different attacks, such as:

1. **Model Inversion Attack (He et al., 2019):** Tries to reverse-engineer the initial input data from the model's output [33]. As DPMix publishes mixed features, this attack can be employed to evaluate its security.
2. **Membership Inference Attack (Shokri et al., 2017):** Tries to discover if the sample belongs to or does not belong to the training set [34]. Attackers train the shadow models to mimic the target model and identify if specific inputs were part of the dataset.

Poorly generalizing models (i.e., the models that fit the training set too tightly) have proven to be more vulnerable to membership inference attacks.

Privacy machine learning is a new field with several promising methods, but each method is subject to its constraints. Differential Privacy is strict but at the expense of utility of the information. Mixup-based methods like DPMix offer new ways to balance privacy preservation with utility preservation and work is ongoing to make it more and more resistant to attacks.

Chapter 3

Methodology

3.1 Basics of Random Mixing

The goal of the differentially private data publishing algorithms is to release a modified dataset so that its privacy can be ensured and still facilitate efficient learning. Differential privacy, introduced by Dwork et al. [7], has been applied as a notion of privacy by a number of research communities. Following this concept, many differentially private data publishing algorithms have been proposed in the literature; however, the majority of them are not suitable for high-dimensional data because their high computational complexity. This necessitates coming up with computationally efficient algorithms that are applicable to high-dimensional data.

In 2019 Kangwook Lee et al. [8] proposed a unique method for differential privacy that can produce synthesized datasets which can be used for both linear and non-linear predictive models.

In a former research work by Karakus et al., it is demonstrated that one could learn a linear model from noiseless mixtures [35]. More recent studies also explored that learning nonlinear models from mixtures is also possible. In [36], Tokozume et al. demonstrated that it was possible to learn a sound recognition model from mixtures of audio waves. Likewise, several recent works indicate that it is possible to train an image classification model using mixtures of images [37]. But Kangwook Lee et al. have proposed an idea that, it is possible to train deep neural networks with noisy mixtures, even the number of degrees is much larger. And they named their method ***Differentially Private Mix (DPMix)*** [8].

3.2 Mathematical Explanation of DPMix Algorithm

In the DPMix algorithm, first, a certain number of samples are chosen randomly from a dataset and then mixed together. The dataset can be of text, image, sound, or others. Let the number of samples mixed together be denoted by ℓ.

Then, the mixture is perturbed with additive Gaussian noise. The resultant dataset is differentially private and guarantees strong privacy while ensuring high accuracies for deep learning models.

3.2.1 Synthetic Data Publishing Algorithm

Consider a dataset where,

- Number of datapoints = n
- Feature matrix $X := [X_1\ X_2\ \cdots\ X_n] \in \mathbb{R}^{d_X \times n}$
- Label matrix $Y := [Y_1\ Y_2\ \cdots\ Y_n] \in \mathbb{R}^{d_Y \times n}$

where (X_i, Y_i) is the i-th data point.

For simplicity in the synthetic data generation process, we assume that X and Y are normalized such that $X \in [0,1]^{d_x \times n}$ and $Y \in [0,1]^{d_y \times n}$. Thus, $\|X_i\| \leq 1$ and $\|Y_i\| \leq 1$ for all $i \in [n]$.

Say, the size of the synthetic dataset to be generated is T.

The synthetic datapoints can be expressed with the following equation:

$$X_t^{'}, Y_t^{'} = (XC_t + Q_t, YC_t + R_t) \quad \forall t \in T$$

Where,

- C_t is the weight vector, $C_t = [C_{t,1}; C_{t,2}; \ldots; C_{t,n}]$
- Here, ℓ entries of C_t are $\frac{1}{\ell}$ and the rest are 0, and the ℓ entries are chosen randomly each time for the generation of each synthetic datapoint.
- Q_t and R_t are white Gaussian noise processes, i.e.,

  $$Q_t \sim \mathcal{N}\left(0, \sigma_x^2 I_{d_x}\right) \quad \text{and} \quad R_t \sim \mathcal{N}\left(0, \sigma_y^2 I_{d_y}\right)$$

  for some (σ_x, σ_y).

For convenience, in this thesis work, I assumed additive noise processes for features and labels both have the same variances, i.e. $\sigma_x = \sigma_y$.

3.2.2 Privacy Parameters

For fixed values of mixture degree ℓ, noise levels (σ_x, σ_y), and number of mixtures T, the DPMix algorithm is (ε, δ)-Differentially Private such that:

$$\varepsilon = \min_{\alpha \in \{2,3,\dots\}} \left(T\varepsilon'_\alpha + \frac{\log(1/\delta)}{\alpha - 1} \right)$$

1. $\varepsilon\alpha'$ Calculation:

$$\varepsilon'_\alpha = \frac{1}{\alpha - 1} \log \left(1 + \left(\frac{l}{n}\right)^2 \binom{\alpha}{2} \min \left(4 \left(\exp\left(\frac{\Delta^2}{l^2}\right) - 1 \right), 2\exp\left(\frac{\Delta^2}{l^2}\right) \right) + 4G(\alpha) \right)$$

2. $G(\alpha)$ Calculation:

$$G(\alpha) = \sum_{j=3}^{\alpha} \left(\frac{l}{n}\right)^j \binom{\alpha}{j} \sqrt{B(2\lfloor j/2 \rfloor) \cdot B(2\lfloor j/2 \rfloor)}$$

3. Bernoulli Polynomial Calculation:

$$B(l) = \sum_{i=0}^{l} (-1)^i \binom{l}{i} \exp\left(\frac{i(i-1)\Delta^2}{2l^2}\right)$$

4. Sensitivity Calculation:

$$\Delta^2 = \frac{d_x}{\sigma_x^2} + \frac{d_y}{\sigma_y^2}$$

Figure 3.1: Necessary formulas to calculate ε

And,

$$\delta = \frac{1}{n}$$

where, n is the total number of samples in the training dataset. [8]

3.3 MNIST Dataset

The MNIST is a famous dataset of handwritten digits (0-9) with labels available on Kaggle.

The dataset consists of two files:

1. `mnist_train.csv`
2. `mnist_test.csv`

The `mnist_train.csv` file contains the 60,000 training examples and labels. The `mnist_test.csv` contains 10,000 test examples and labels. Each row consists of 785 values: **the first value is the label value (a number from 0 to 9)** and **the remaining 784 values are the pixel values (a number from 0 to 255).**

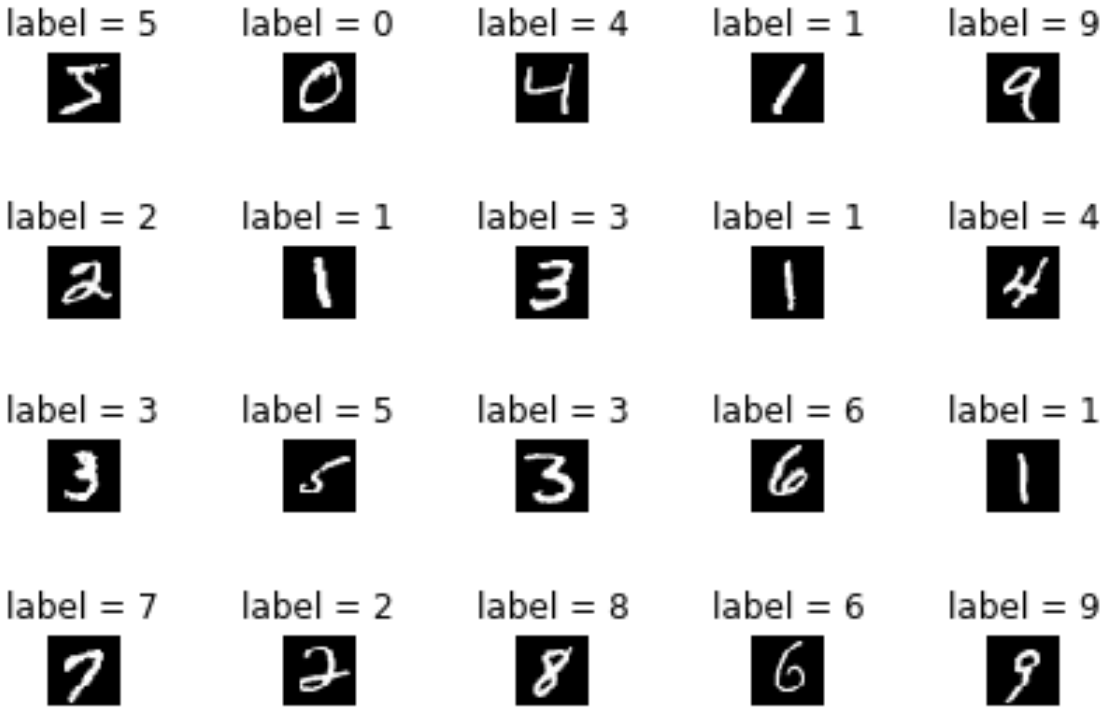


Figure 3.2: Some image samples of handwritten digits from MNIST dataset along with their corresponding labels

The MNIST dataset contains grayscale images of handwritten digits, where each image is of size 28×28 (total 784 pixels in each image). The darkest pixel is represented by 0, and the brightest pixel has a value of 255.

In this thesis, I used the MNIST dataset from Kaggle for implementation and analysis of the DPMix algorithm.

3.4 Implementation of DPMix on MNIST

The training dataset of MNIST has 60,000 samples. So,

$$n = 60000$$

The dimension of features for each sample, $d_x = 784$.

The labels (0-9) were one-hot encoded before perturbing with noise, so the dimension of labels, $d_y = 10$.

3.4.1 Steps of Synthesizing Differentially Private Dataset

1. First, choose a certain value of T – the number of synthetic samples to be generated.
2. Choose a certain value of ℓ – the number of original samples to be mixed.
3. Choose the values of σ_x and σ_y – the standard deviations of the noises to be added with the mixtures. (For simplicity, we will assume $\sigma_x = \sigma_y$ in this case.)
4. For convenience, scale the feature matrix X within $[0, 1]$ and one-hot encode the label matrix Y.
5. To generate each synthetic sample, follow the steps:
   (a) Generate the weight vector C_t. C_t is an $n \times 1$ matrix, where ℓ randomly chosen elements have a value of $1/\ell$. The rest are zero.
   (b) Generate Gaussian noise vectors Q_t and R_t with mean zero and standard deviation σ_x (or σ_y).
   (c) Generate a synthetic datapoint $(X_t^{'}, Y_t^{'})$ using the following formulas:

$$X_t^{'} = X^T \times C_t + Q_t$$

$$Y_t^{'} = Y^T \times C_t + R_t$$

   where X is the feature matrix of the MNIST train dataset with dimension $d_x \times n$, and Y is the label matrix of the MNIST train dataset with dimension $d_y \times n$ (after one-hot encoding).
   (d) Repeat this process for $t = [1, T]$.
6. And thus, we get a synthetic dataset of T samples that is differentially private and can be used for training deep-learning models.

3.5 Analyzing the Effects of Various Parameters on the Accuracy of DPMix Algorithm

We can determine the effects of various parameters like noise (σ_x), mixture degree (ℓ), and the size of the synthetic dataset (T) on the performance of the DPMix algorithm in training deep learning models.

For this, we take a slightly different approach. Instead of randomly mixing images of random digits from the dataset, we will mix only the images having the same labels. Meaning, we will only mix images of zeros with one another to produce an image of a synthetic zero. Similarly, for all other digits (1-9), we will produce synthetic samples. Then both the images produced from mixtures and their one-hot encoded labels will be perturbed with additive Gaussian noise. This way, we will get our desired synthetic dataset that is differentially private.

Then, we will train a deep learning model with the synthetic dataset and measure the accuracy by evaluating the model on the original MNIST dataset. We can vary the parameters σ_x, ℓ, and T to see how the accuracy of deep learning models changes with the changing of these parameters.

It is obvious that an increase in the value of the noise parameter (σ_x) will decrease the test accuracy, while an increase in the value of T will increase the accuracy.

It is possible to show through a plot that there exists a sweet spot on ℓ that maximizes the prediction accuracy given a required privacy level. (Both increasing or decreasing the value of ℓ from that optimal value will cause a decrease in the prediction accuracy).

Chapter 4

Results and Discussion

4.1 Implementing DPMix Algorithm with MNIST Dataset

The MNIST dataset has 60,000 training samples and 10,000 testing samples. So, we have the following parameters:

$$d_x = 784$$

$$d_y = 10$$

$$n = 60000$$

If we want to generate a synthetic dataset of 10,000 samples with a mixture degree of 256, we have:

$$T = 10000$$

$$\ell = 256$$

And, we want the synthetic dataset to be (ε, δ)-differentially private such that:

$$\varepsilon = 15$$

which results in $\sigma_x = \sigma_y = 0.0711$.

And,

$$\delta = \frac{1}{n} = \frac{1}{60000}$$

After generating the synthetic dataset using the formulas mentioned in Section 3.4, we train a CNN model with the newly generated private dataset.

The CNN model is built on TensorFlow.

Layer Type	**Configuration**
Conv2d	32 filters of 5×5, activation ReLU
Max-Pooling	2×2
Conv2d	48 filters of 5×5, activation ReLU
Max-Pooling	2×2
Fully connected	100 units, activation ReLU
Fully connected	100 units, activation ReLU
Fully connected	10 units, activation Softmax

Table 4.1: Structure of the CNN model

The model was trained with the synthetic dataset for 50 epochs. The initial learning rate was 0.0001. We halved the learning rate after each 10 epochs during the training session. The validation split was 10%.

The Adam optimizer was used.

Besides, I set all the negative values in the synthetic feature and label matrices to 0.

Instead of reverse one-hot encoding of the noisy labels using the `argmax()` function, the noisy labels were used for training. This enables the label smoothing advantages for training a CNN model, such as preventing overfitting and improving generalization, etc.

As I used soft labels instead of hard one-hot encoded labels for training, I used the KL Divergence as the loss function. It allows the model to learn from soft labels without forcing it to commit to a single class and helps the CNN to generalize better rather than becoming overconfident in a specific class. [38]

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 1s 1ms/step - accuracy: 0.6545 - loss: 1.7592
Test accuracy on original data: 0.6834999918937683
```

Figure 4.1: DPMix Prediction Accuracy

With all these constraints above, I got a prediction accuracy of nearly 68.35% when evaluated on the MNIST test dataset.

4.2 Performance of DPMix Algorithm Based on Various Parameters

My next target was to observe how the variation in parameters σ_x, ℓ, and T changes the accuracy of the predictive model.

For this, instead of randomly mixing samples from all digits, I mixed only the samples that have the same label value, i.e., zeros were mixed together to generate some synthetic samples, ones were mixed together to generate some synthetic samples, and so on.

```python
import pandas as pd
import numpy as np

# Load the CSV file containing MNIST training data
train_data = pd.read_csv('mnist_train.csv')

# The first column is the label (digit), and the rest are pixel values
train_labels = train_data.iloc[:, 0].values  # First column (labels)
train_images = train_data.iloc[:, 1:].values  # All other columns (pixel values)

# Count the occurrences of each digit in the training labels
digit_counts = np.bincount(train_labels)

# Print the number of images for each digit
for i in range(10):
    print(f"Number of {i}s in the MNIST training dataset: {digit_counts[i]}")
```

Executed at 2024.12.01 21:51:54 in 1s 555ms

```
Number of 0s in the MNIST training dataset: 5923
Number of 1s in the MNIST training dataset: 6742
Number of 2s in the MNIST training dataset: 5958
Number of 3s in the MNIST training dataset: 6131
Number of 4s in the MNIST training dataset: 5842
Number of 5s in the MNIST training dataset: 5421
Number of 6s in the MNIST training dataset: 5918
Number of 7s in the MNIST training dataset: 6265
Number of 8s in the MNIST training dataset: 5851
Number of 9s in the MNIST training dataset: 5949
```

Figure 4.2: Number of samples calculated for each digit in MNIST training dataset

The number of samples for each digit in the MNIST training dataset is close to 6000. For convenience, we will take $n = 6000$ for calculations.

4.2.1 Accuracy vs. Noise Plot

For the following parameters:

$$d_x = 784 \quad \text{(MNIST dataset feature dimension)}$$

$$d_y = 10 \quad \text{(MNIST dataset label dimension, after one-hot encoding)}$$

$$\ell = 128 \quad \text{(Number of random samples to mix)}$$

$$n = 6000 \quad \text{(Number of samples in the dataset)}$$

$$\text{Number of synthetic samples generated for each digit} = 1200$$

$$T = 10 \times 1200 = 12000 \quad \text{(Total number of synthetic data points to be generated)}$$

$$\delta = \frac{1}{n} \quad \text{(Privacy budget parameter)}$$

I varied the σ_x value from 0.2 to 3.0 with 0.1 increments. For each value of σ_x, 10 synthetic datasets were generated using the DPMix algorithm, and I trained the CNN models with those 10 synthetic datasets. After each training, the model accuracy was evaluated on the MNIST test dataset. Then, I took the average of the 10 accuracy values to get a mean accuracy value for the σ_x on the scene.

This time, instead of using noisy labels for training, I reverse one-hot encoded the noisy labels with the `argmax()` function. So, this time, Categorical Cross-Entropy was used as the loss function, since CCE works better on hard one-hot encoded labels instead of distributed (noisy) soft one-hot encoded labels.

Finally, it was possible to plot the Accuracy vs. Noise (σ_x). Also, the privacy parameter ε, which largely depends on the noise parameters (σ_x, σ_y), was plotted on the second y-axis.

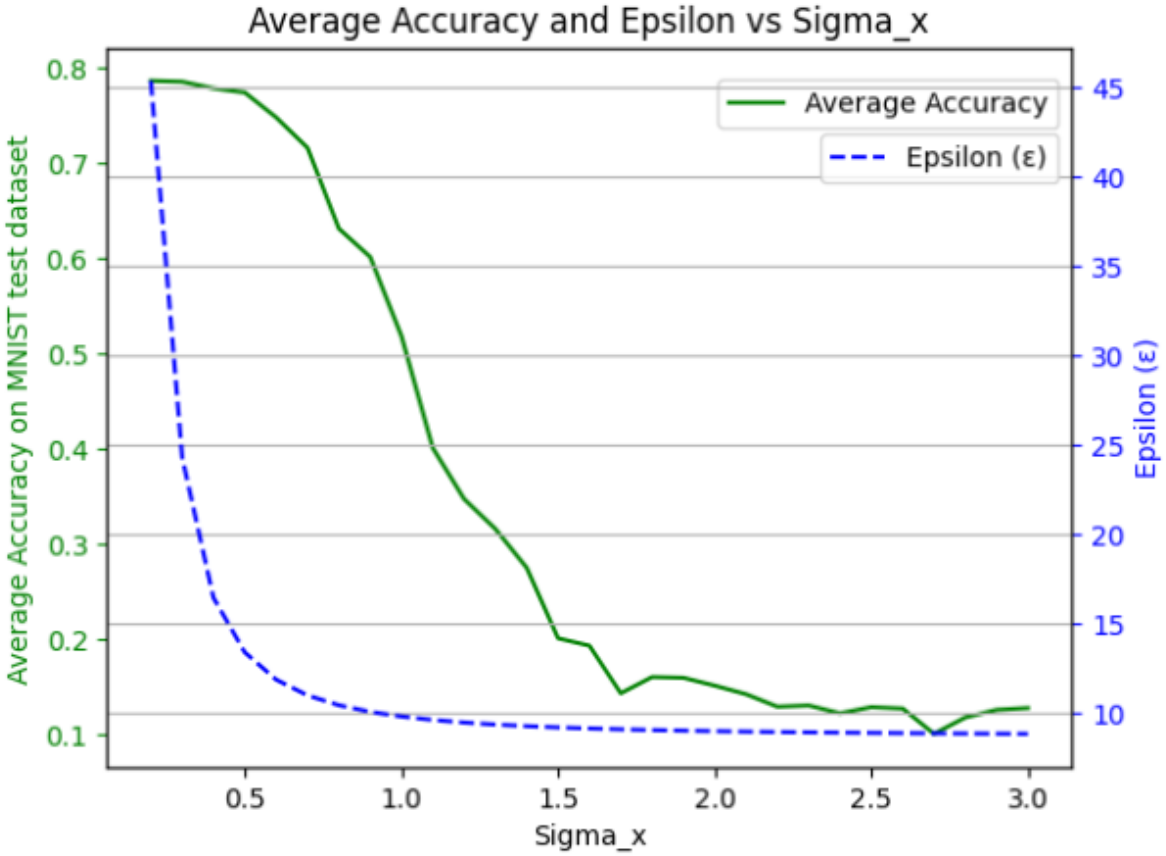


Figure 4.3: Accuracy vs. Noise vs. ε plot

From the plot, it is evident that accuracy decreases as the value of noise increases, and so does the value of ε.

- Lower ε represents the presence of higher additive noise and vice versa.
- Lower ε = Strong privacy
- Higher ε = Weak privacy

4.2.2 Accuracy vs. ℓ Plot

This time, I varied the value of the mixture degree ℓ to see how it affects the predictive accuracy.

For a fixed value of $\sigma_x = 0.7$ $(= \sigma_y)$, we get the following plot:

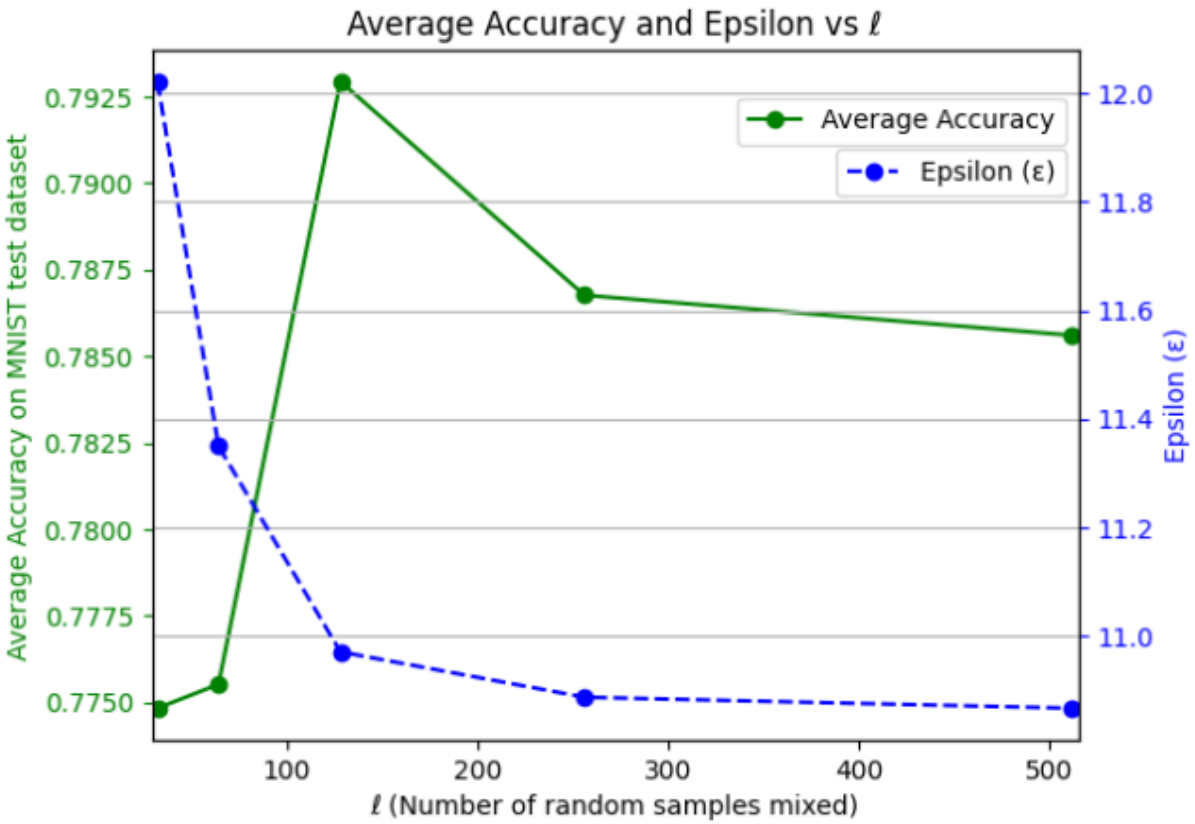


Figure 4.4: Accuracy vs. ℓ vs. ε plot

ℓ	Accuracy
32	77.50%
64	77.60%
128	79.30%
256	78.70%
512	78.60%

Table 4.2: Variation of accuracy with respect to ℓ.

As mentioned in the research paper by Lee et al. [8], since it's a non-linear model, we got a sweet spot on ℓ where accuracy is maximum, i.e., the model performs best. In this case, it is $\ell^* = 128$.

4.2.3 Accuracy vs. T Plot

For a fixed value of $\sigma_x = 0.7$ and $\ell = 128$, we get the accuracy vs. T curve as follows:

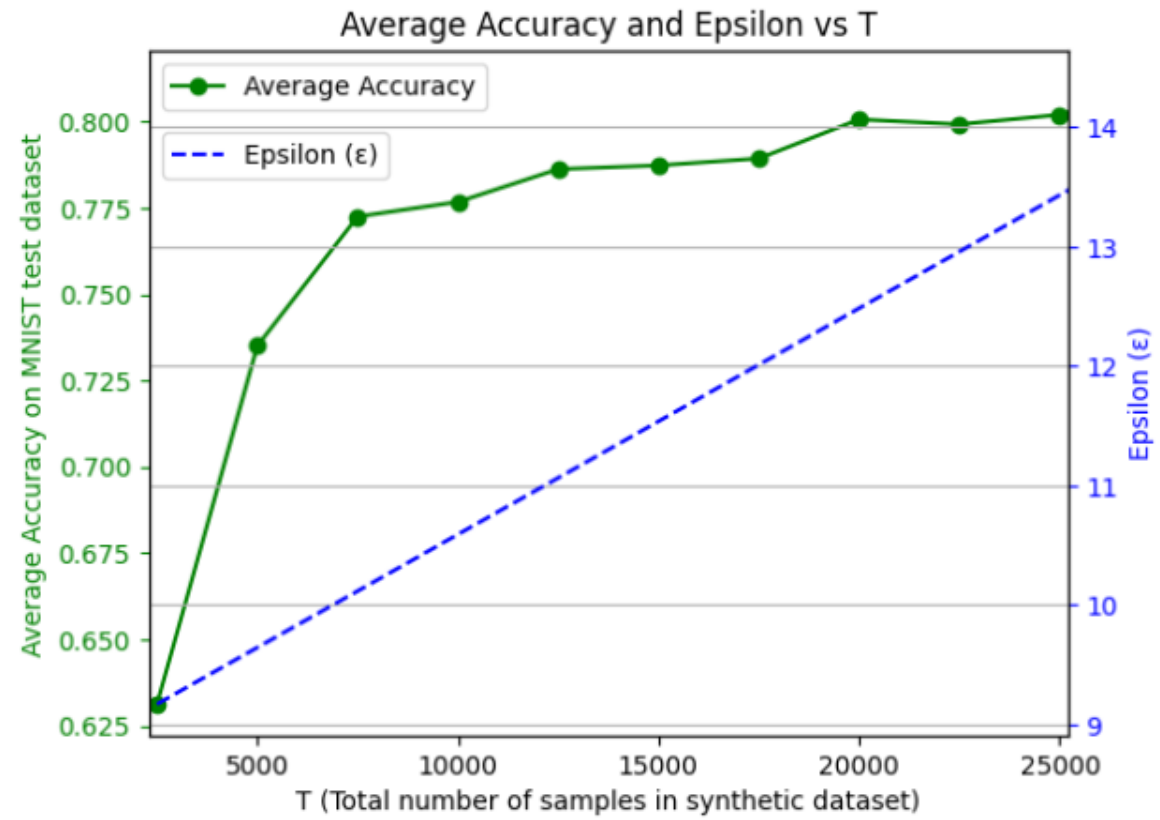


Figure 4.5: Accuracy vs. T vs. ε plot.

From the plot, it is evident that as the number of samples in the synthetic dataset increases, the predictive model performs better and better.

Also, the equation used to calculate ε was:

$$\varepsilon = \min_{\alpha \in \{2,3,\ldots\}} \left(T\varepsilon'_\alpha + \frac{\log(1/\delta)}{\alpha - 1} \right)$$

Here, in this equation, on the RHS, the 2$^{\text{nd}}$ term is small compared to the 1$^{\text{st}}$ term. And in the first term $T\varepsilon'_\alpha$, the ε'_α is constant for a fixed value of σ_x and ℓ. So, we see the ε plot is varying in a nearly linear fashion corresponding to the variation of T.

Chapter 5

Evaluation

5.1 Conclusion

It was shown in this research work that the DPMix algorithm guarantees strong differential privacy for large datasets while ensuring efficient learning with the synthetic datasets. It can provide reasonably high security for a small additive noise. It is possible to train both linear and non-linear models with the datasets generated through the DPMix algorithm. With careful inspection, we can find the optimal value of mixture degree ℓ for which the deep learning models perform best with minimal additive noise. Overall, this technique is new and uniquely handles the data manipulation process compared to the existing methods. Thus, DPMix can be an excellent choice for tech companies, hospitals, and other entertainment industries to effectively secure user data, since instead of using the original data, they can now use the synthetic data to improve their productivity and carry out related research works.

5.2 Future Work

DPMix is one of the best methodologies invented for the privacy protection of large databases to this day. However, there is still room for improvement in many areas. Until now, this technique has been applied only to static visual datasets. It should be checked how it performs on other types of datasets, such as video, sound, etc.

More inquiry should be done to add new features to the algorithm so that it can perform better. Further mathematical analysis should be carried out to find the flaws (if there are any) in this algorithm, and this can help to overcome the existing shortcomings. Giant tech and AI companies can adopt this method and use it for data privacy while improving its performance and efficiency more and more through their experts and engineers.

Bibliography

[1] A. Narayanan and V. Shmatikov, *Robust De-anonymization of Large Sparse Datasets*, in *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, May 2008, pp. 111–125. doi: 10.1109/SP.2008.33.

[2] Simply Explained, *Differential Privacy - Simply Explained*, YouTube, Mar. 15, 2018. [Online]. Available: https://www.youtube.com/watch?v=gI0wk1CXlsQ.

[3] L. Sweeney, *Computational Disclosure Control - A Primer on Data Privacy Protection*, Ph.D. dissertation, Massachusetts Institute of Technology, Dept. of Electrical Engineering and Computer Science, Cambridge, MA, USA, 2001. [Online]. Available: http://groups.csail.mit.edu/mac/classes/6.805/articles/privacy/sweeney-thesis-draft.pdf.

[4] *Differential privacy*, Wikipedia, The Free Encyclopedia. [Online]. Available: https://en.wikipedia.org/wiki/Differential_privacy.

[5] C. Dwork, F. McSherry, K. Nissim, and A. Smith, *Calibrating Noise to Sensitivity in Private Data Analysis*, *Journal of Privacy and Confidentiality*, vol. 7, no. 3, pp. 17-51, May 30, 2017. doi: 10.29012/jpc.v7i3.405.

[6] M. Hilton and C. Cal, *Differential Privacy: A Historical Survey*, in *Proceedings of the Conference/Workshop*, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:16861132.

[7] C. Dwork, *Differential Privacy: A Survey of Results*, in *Theory and Applications of Models of Computation*, M. Agrawal, D. Du, Z. Duan, and A. Li, Eds. Berlin, Germany: Springer, 2008, vol. 4978, pp. 1–19. doi: 10.1007/978-3-540-79228-4_1.

[8] K. Lee, H. Kim, K. Lee, and C. Suh, *Synthesizing Differentially Private Datasets using Random Mixing*, in *2019 IEEE International Symposium on Information Theory (ISIT)*, Paris, France, Jul. 2019, pp. 1041–1045. doi: 10.1109/ISIT.2019.8849381.

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, *Communication-efficient learning of deep networks from decentralized data*, in *Proc. 20th Int. Conf. on Artificial Intelligence and Statistics*, PMLR, 2017, vol. 54, pp. 1273–1282.

[10] L. Zhu and S. Han, *Deep leakage from gradients*, in *Federated Learning*, Q. Yang, L. Fan, and H. Yu, Eds. Cham, Switzerland: Springer, 2020, vol. 12500, pp. 17–26. doi: 10.1007/978-3-030-63076-8_2.

[11] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, *Split learning for health: Distributed deep learning without sharing raw patient data*, *arXiv preprint*, arXiv:1812.00564, 2018.

[12] Z. He, T. Zhang, and R. B. Lee, *Model inversion attacks against collaborative inference*, in *Proc. 35th Annu. Computer Security Applications Conf. (ACSAC '19)*, San Juan, Puerto Rico, USA, 2019, pp. 148–162. doi: 10.1145/3359789.3359824.

[13] C. Dwork and A. Roth, *The algorithmic foundations of differential privacy*, *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014. doi: 10.1561/0400000042.

[14] Ú. Erlingsson, V. Pihur, and A. Korolova, *RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response*, in *Proc. 2014 ACM SIGSAC Conf. on Computer and Communications Security (CCS '14)*, Scottsdale, AZ, USA, 2014, pp. 1054–1067. doi: 10.1145/2660267.2660348.

[15] K. Kenthapadi and T. T. L. Tran, *PriPeARL: A framework for privacy-preserving analytics and reporting at LinkedIn*, *Proc. 27th ACM Int. Conf. on Information and Knowledge Management*, 2018.

[16] Differential Privacy Team, *Learning with privacy at scale*, *Apple Machine Learning Research*, Dec. 6, 2017.

[17] B. Ding, J. Kulkarni, and S. Yekhanin, *Collecting telemetry data privately*, in *Proc. Advances in Neural Information Processing Systems 30 (NIPS 2017)*, 2017.

[18] J. Zhang, G. Cormode, C. Procopiuc, D. Srivastava, and X. Xiao, *PrivBayes: Private data release via Bayesian networks*, *ACM Trans. Database Syst.*, vol. 42, no. 4, p. 25, 2017. doi: 10.1145/3134428.

[19] C. Xu, J. Ren, Y. Zhang, Z. Qin, and K. Ren, *DPPro: Differentially private high-dimensional data release via random projection*, *IEEE Trans. Inf. Forensics Security*, vol. 12, no. 12, pp. 3081–3093, 2017.

[20] L. Xie, K. Lin, S. Wang, F. Wang, and J. Zhou, *Differentially private generative adversarial network*, *arXiv preprint*, arXiv:1802.06739, 2018.

[21] X. Zhang, S. Ji, and T. Wang, *Differentially private releasing via deep generative model*, *arXiv preprint*, arXiv:1801.01594, 2018.

[22] S. Takagi, T. Takahashi, Y. Cao, and M. Yoshikawa, *P3GM: Private high-dimensional data release via privacy preserving phased generative model*, in *Proc. 2021 IEEE 37th Int. Conf. on Data Engineering (ICDE)*, pp. 169–180, 2021.

[23] C. Dwork, F. McSherry, K. Nissim, and A. Smith, *Calibrating noise to sensitivity in private data analysis*, in *Theory of Cryptography*, S. Halevi and T. Rabin, Eds. Berlin, Heidelberg: Springer, 2006, vol. 3876, Lecture Notes in Computer Science, pp. 265–284. doi: 10.1007/11681878_14.

[24] I. Mironov, *Rényi differential privacy*, in *Proc. 2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017, pp. 263–275.

[25] J. Dong, A. Roth, and W. Su, *Gaussian differential privacy*, *Journal of the Royal Statistical Society*, 2021.

[26] R. McKenna, G. Miklau, and D. Sheldon, *Winning the NIST contest: A scalable and general approach to differentially private synthetic data*, *CoRR*, vol. abs/2108.04978, 2021.

[27] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, *Deep learning with differential privacy*, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 308–318, 2016, doi: 10.1145/2976749.2978318.

[28] F. Tramer and D. Boneh, *Differentially private learning needs better features (or much more data)*, *International Conference on Learning Representations*, 2021.

[29] H. Zhang, M. Cissé, Y. Dauphin, and D. Lopez-Paz, *mixup: Beyond Empirical Risk Minimization*, *arXiv*, 2017.

[30] M. Chidambaram, X. Wang, Y. Hu, C. Wu, and R. Ge, *Towards Understanding the Data Dependency of Mixup-style Training*, *arXiv*, 2021.

[31] Y. Huang, Z. Song, K. Li, and S. Arora, *InstaHide: Instance-hiding schemes for private distributed learning*, *Proceedings of the 37th International Conference on Machine Learning (ICML 2020)*, pp. 4457–4468, 2020.

[32] N. Carlini, S. Deng, S. Garg, S. Jha, S. Mahloujifar, M. Mahmoody, S. Song, A. Thakurta, and F. Tramer, *An attack on InstaHide: Is private learning possible with instance encoding?*, *arXiv preprint arXiv:2011.05315*, 2020.

[33] Z. He, T. Zhang, and R. B. Lee, *Model inversion attacks against collaborative inference*, *Proceedings of the 35th Annual Computer Security Applications Conference*, San Juan, Puerto Rico, USA, 2019, pp. 148–162, doi: 10.1145/3359789.3359824.

[34] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, *Membership inference attacks against machine learning models*, *2017 IEEE Symposium on Security and Privacy*, San Jose, CA, USA, 2017, pp. 3–18, doi: 10.1109/SP.2017.41.

[35] C. Karakus, Y. Sun, S. Diggavi, and W. Yin, *Straggler mitigation in distributed optimization through data encoding*, *Advances in Neural Information Processing Systems*, 2017, pp. 5434–5442.

[36] Y. Tokozume, Y. Ushiku, and T. Harada, *Learning from between-class examples for deep sound recognition*, in *International Conference on Learning Representations (ICLR)*, Apr. 2018.

[37] H. Inoue, *Data augmentation by pairing samples for image classification*, *arXiv:1801.02929*, Jan. 2018.

[38] D. Li, Y. Cao, and Y. Yao, *Optimizing Random Mixup with Gaussian Differential Privacy*, Feb. 15, 2022. [Online]. Available: arXiv:2202.06467.

Appendix A

Python Code for DPMix Algorithm

A.1 Generating Synthetic Dataset

```
# Function for generating synthetic dataset
def generate_synthetic_data(X, y_one_hot, T, l, sigma_x, sigma_y):
    n, d_X = X.shape  # n: number of original data points, d_X: dimensionality of feature space
    d_y = y_one_hot.shape[1]  # Dimensionality of the one-hot encoded labels
    X_synthetic = np.zeros((T, d_X))  # Initialize the synthetic feature matrix
    y_synthetic = np.zeros((T, d_y))  # Initialize the synthetic label matrix

    for t in range(T):
        # Step 1: Select l data points randomly without replacement
        indices = np.random.choice(n, l, replace=False)

        # Step 2: Create the weight vector for the mixture
        C_t = np.zeros(n)  # Initialize the weight vector with zeros
        C_t[indices] = 1 / l  # Set the weights of the selected data points to 1 / l

        # Step 3: Add Gaussian noise for differential privacy
        Q_t = np.random.normal(0, sigma_x, d_X)  # Gaussian noise for features
        R_t = np.random.normal(0, sigma_y, d_y)  # Gaussian noise for labels
```

```
19
20          # Step 4: Generate the synthetic data point
21          X_synthetic[t] = X.T @ C_t + Q_t  # Weighted sum of selected
    features plus noise
22          y_synthetic[t] = y_one_hot.T @ C_t + R_t  # Weighted sum of
    selected one-hot labels plus noise
23
24      return X_synthetic, y_synthetic
25
26 # Generate the synthetic dataset
27 X_synthetic, y_synthetic = generate_synthetic_data(X_train,
    y_train_one_hot, T, l, sigma_x, sigma_y)
```

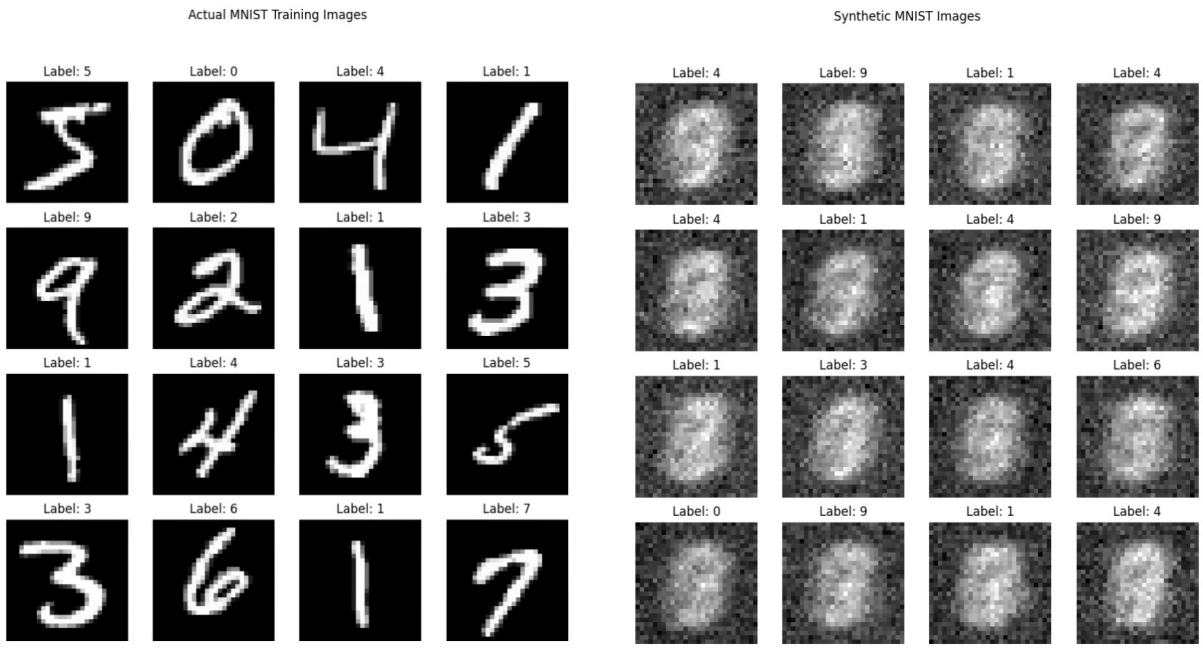


Figure A.1: Some samples from MNIST dataset and Synthetic dataset

A.2 CNN Model

```
1 # CNN model
2 def build_cnn_model():
3     model = models.Sequential()
4     model.add(Input(shape=(28, 28, 1)))
5     model.add(layers.Conv2D(32, (5, 5), activation='relu'))
6     model.add(layers.MaxPooling2D((2, 2)))
7     model.add(layers.Conv2D(48, (5, 5), activation='relu'))
```

```
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Flatten())
    model.add(layers.Dense(100, activation='relu'))
    model.add(layers.Dense(100, activation='relu'))
    model.add(layers.Dense(10, activation='softmax'))
    return model
```

A.3 Epsilon Calculation

```
import numpy as np
import math
from scipy.special import comb
from scipy.linalg import sqrtm

# Parameters
dx = 784  # MNIST dataset feature dimension
dy = 10  # MNIST dataset label dimension (one-hot encoded)
l = 128  # Mixture degree
n = 60000  # Number of samples in the dataset
sigma_x = 0.1132  # Standard deviation for features (example value)
sigma_y = sigma_x  # Standard deviation for labels (same as sigma_x)
T = 10000  # Number of synthetic data points to generate

# Sensitivity calculation
Delta2 = (dx / sigma_x**2) + (dy / sigma_y**2)

# Bernoulli polynomial B(l) calculation
def B(l, Delta2):
    sum_B = 0
    for i in range(l + 1):
        sum_B += (-1)**i * comb(l, i) * np.exp(i * (i - 1) * Delta2
   / (2 * l**2))
    return sum_B

# G(alpha) calculation
def G(alpha, l, n, Delta2):
    sum_G = 0
    for j in range(3, alpha + 1):
        sum_G += (l / n)**j * comb(alpha, j) * math.sqrt(B(2 * (j //
   2), Delta2) * B(2 * (j - j // 2), Delta2))
```

```
    return sum_G

#     ' calculation
def epsilon_alpha_prime(alpha, l, n, Delta2):
    min_term = min(4 * (np.exp(Delta2 / l**2) - 1), 2 *
   np.exp(Delta2 / l**2))
    term1 = (l / n)**2 * comb(alpha, 2) * min_term
    term2 = 4 * G(alpha, l, n, Delta2)
    return (1 / (alpha - 1)) * np.log(1 + term1 + term2)

# Final    calculation
def epsilon(T, l, n, Delta2, delta=1/n):
    epsilons = []
    for alpha in range(2, 101):  # Consider    values up to 100
        try:
            epsilon_alpha = epsilon_alpha_prime(alpha, l, n, Delta2)
            epsilon_total = T * epsilon_alpha + np.log(1 / delta) /
   (alpha - 1)
            epsilons.append(epsilon_total)
        except (OverflowError, ValueError):
            break  # Stop if an error occurs to avoid instability
    return min(epsilons)

# Calculate
epsilon_value = epsilon(T, l, n, Delta2)
print(f"Calculated   : {epsilon_value}")
```

Appendix B

Python Code for Performance Analysis of DPMix

B.1 Accuracy vs Noise

```
# Generate 10 synthetic datasets using the DPMix algorithm
def generate_10_synthetic_datasets(X, y, sigma_x, sigma_y, l, num_samples_per_class):
    num_classes = 10
    datasets = []  # List to store the 10 synthetic datasets

    for dataset_idx in range(10):  # Loop to generate 10 datasets
        X_synthetic = []
        y_synthetic = []

        for digit in range(num_classes):
            class_indices = np.where(y == digit)[0]
            sampled_indices = np.random.choice(class_indices, l, replace=False)
            X_class = X[sampled_indices]

            original_label = np.eye(num_classes)[digit]  # One-hot encoding of the digit

            for _ in range(num_samples_per_class):
                weights = np.ones(l) / l  # Equal weights of 1/l for each sample
                mixed_sample = np.tensordot(weights, X_class, axes=1)
```

```
                noise_x = np.random.normal(0, sigma_x,
    mixed_sample.shape)
                noisy_sample = mixed_sample + noise_x

                noise_y = np.random.normal(0, sigma_y,
    original_label.shape)  # Gaussian noise for labels
                noisy_label = original_label + noise_y

                # Reverse one-hot encoding: label is the index of
    the max value
                label_from_noisy = np.argmax(noisy_label)

                X_synthetic.append(noisy_sample)
                y_synthetic.append(label_from_noisy)

        X_synthetic = np.array(X_synthetic)
        y_synthetic = np.array(y_synthetic)

        # Ensure no negative values in synthetic images and scale
    within [0, 1]
        X_synthetic[X_synthetic < 0] = 0  # Setting all negative
    values to 0
        X_synthetic = X_synthetic.reshape(-1, 28, 28, 1)  # Reshape
    back to image format

        # Append the synthetic dataset to the list
        datasets.append((X_synthetic, y_synthetic))

    return datasets  # Return a list of 10 synthetic datasets

# Initialize lists for accuracies
synthetic_accuracies = []
test_accuracies = []

# Display original images once
print(f"Displaying original images from MNIST dataset")
display_original_images(x_train, y_train)

# Initialize an empty list to store average accuracies
average_accuracies = []

```

```
# Loop over sigma_x, sigma_y values
for sigma_x, sigma_y in zip(sigma_x_values, sigma_y_values):
    print(f"Generating 10 synthetic datasets for sigma_x = {sigma_x} and sigma_y = {sigma_y}")

    # Generate 10 synthetic datasets for current sigma_x and sigma_y
    synthetic_datasets = generate_10_synthetic_datasets(x_train, y_train, sigma_x, sigma_y, l, num_synthetic_samples_per_class)

    # Initialize a list to store test accuracies for this (sigma_x, sigma_y)
    test_accuracies_per_sigma = []

    for dataset_idx, (x_synthetic, y_synthetic) in enumerate(synthetic_datasets):
        # Display synthetic images for the first dataset
        if dataset_idx == 0:
            print(f"Displaying synthetic images for sigma_x = {sigma_x}, sigma_y = {sigma_y}, dataset {dataset_idx + 1}")
            display_synthetic_images(x_synthetic, y_synthetic)

        # One-hot encode synthetic labels
        y_synthetic_one_hot = np.eye(10)[y_synthetic]

        # Split synthetic data into training and validation sets
        x_synth_train, x_synth_val, y_synth_train, y_synth_val = train_test_split(
            x_synthetic, y_synthetic_one_hot, test_size=validation_split_ratio)

        print(f"Training model on synthetic dataset {dataset_idx + 1} for sigma_x = {sigma_x} and sigma_y = {sigma_y}")

        # Build the CNN model
        model = cnn_model()

        # Add callbacks for early stopping and model checkpointing
        early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)
        model_checkpoint = ModelCheckpoint(
```

```
86
    f'best_model_sigma_{sigma_x}_{sigma_y}_dataset_{dataset_idx +
    1}.keras',
87              save_best_only=True, monitor='val_accuracy', mode='max')
88
89          # Train the model with train/validation split
90          model.fit(
91              x_synth_train, y_synth_train,
92              validation_data=(x_synth_val, y_synth_val),
93              epochs=epochs, callbacks=[early_stopping,
    model_checkpoint], verbose=1)
94
95          # Evaluate on original MNIST test dataset
96          test_loss, test_acc = model.evaluate(x_test, y_test_one_hot,
    verbose=0)
97          print(f"Test accuracy for dataset {dataset_idx + 1}:
    {test_acc:.4f}")
98          test_accuracies_per_sigma.append(test_acc)
99
100      # Compute and store average accuracy for the current sigma_x and
    sigma_y
101      avg_accuracy = np.mean(test_accuracies_per_sigma)
102      print(f"Average test accuracy for sigma_x = {sigma_x}, sigma_y =
    {sigma_y}: {avg_accuracy:.4f}")
103      average_accuracies.append(avg_accuracy)
```

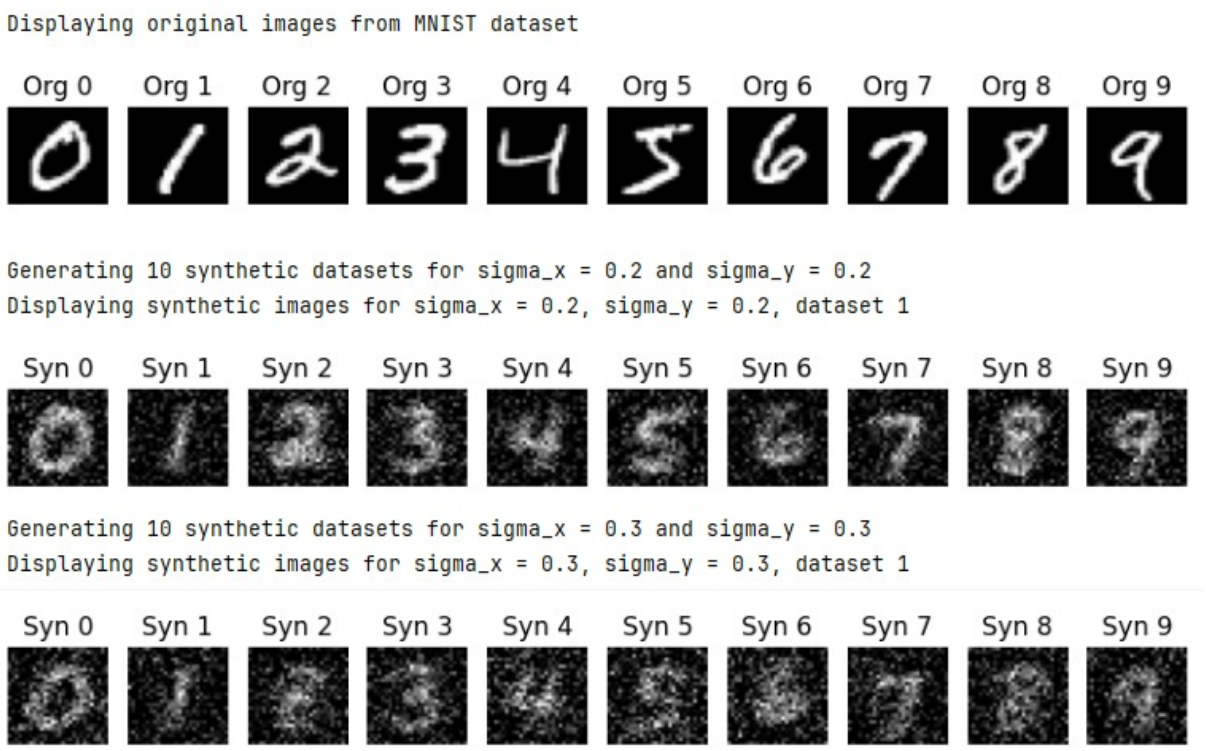


Figure B.1: Some samples from MNIST dataset and Synthetic datasets with varying noise

B.2 Accuracy vs Mixture Degree

```python
# Loop within l_values
for l in l_values:
    print(f"Generating synthetic datasets for l = {l}")

    # Generate 10 synthetic datasets for l
    synthetic_datasets = generate_synthetic_datasets(x_train, y_train, sigma_x, sigma_x, l, num_synthetic_samples_per_class)

    # Initialize a list to store test accuracies for this l
    test_accuracies_per_l = []

    for dataset_idx, (x_synthetic, y_synthetic) in enumerate(synthetic_datasets):
        # Display synthetic images for the first dataset
        if dataset_idx == 0:
            print(f"Displaying synthetic images for l = {l}, dataset {dataset_idx+1}")
            display_synthetic_images(x_synthetic, y_synthetic)
```

```
16
17        # One-hot encode synthetic labels
18        y_synthetic_one_hot = np.eye(10)[y_synthetic]
19
20        # Split synthetic data into training and validation sets
21        x_synth_train, x_synth_val, y_synth_train, y_synth_val =
    train_test_split(
22            x_synthetic, y_synthetic_one_hot,
    test_size=validation_split_ratio)
23
24        print(f"Training model on synthetic dataset {dataset_idx +
    1} for l = {l}")
25
26        # Build the CNN model
27        model = cnn_model()
28
29        # Add callbacks for early stopping and model checkpointing
30        early_stopping = EarlyStopping(monitor='val_accuracy',
    patience=3, restore_best_weights=True)
31        model_checkpoint = ModelCheckpoint(
32            f'best_model_l={l}_dataset_{dataset_idx + 1}.keras',
33            save_best_only=True, monitor='val_accuracy', mode='max')
34
35        # Train the model with train/validation split
36        model.fit(
37            x_synth_train, y_synth_train,
38            validation_data=(x_synth_val, y_synth_val),
39            epochs=epochs,
40            verbose=1,
41            callbacks=[early_stopping, model_checkpoint]  # Include
    callbacks here
42        )
43
44        # Evaluate on original MNIST test dataset
45        test_loss, test_acc = model.evaluate(x_test, y_test_one_hot,
    verbose=0)
46        test_accuracies_per_l.append(test_acc)
47
48     avg_acc = np.mean(test_accuracies_per_l)
49     print(f"Average test accuracy for l = {l}: {avg_acc:.4f}")
50     average_accuracies.append(avg_acc)
```

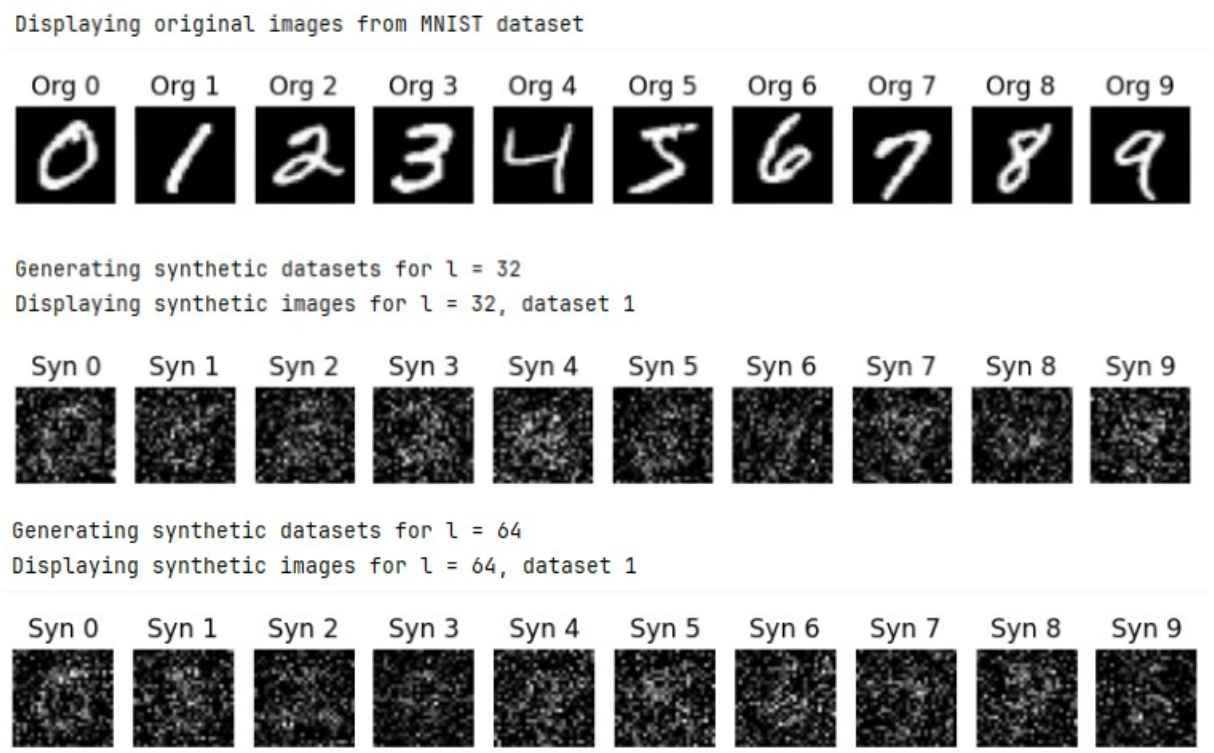


Figure B.2: Some samples from MNIST dataset and Synthetic datasets with varying ℓ

B.3 Accuracy vs Size of Synthetic Dataset

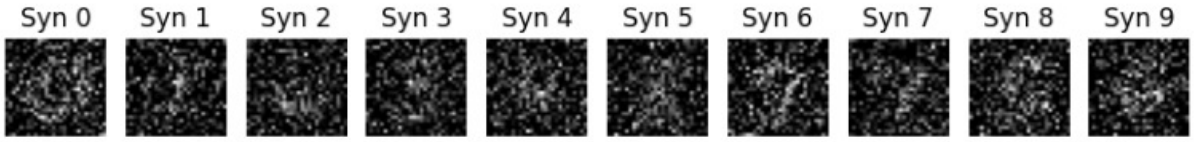
```
# Loop over num_samples_per_class values
for num_samples in num_samples_per_class_list:
    print(f"Generating synthetic datasets for T = {10*num_samples}")

    # Generate 10 synthetic datasets for num_samples
    synthetic_datasets = generate_synthetic_datasets(x_train,
   y_train, sigma_x, sigma_x, l, num_samples)

    # Initialize a list to store test accuracies for this num_samples
    test_accuracies_per_T = []

    for dataset_idx, (x_synthetic, y_synthetic) in
   enumerate(synthetic_datasets):
        # Display synthetic images for the first dataset
        if dataset_idx == 0:
            print(f"Displaying synthetic images for T =
   {10*num_samples}, dataset {dataset_idx+1}")
            display_synthetic_images(x_synthetic, y_synthetic)
```

```
16
17        # One-hot encode synthetic labels
18        y_synthetic_one_hot = np.eye(10)[y_synthetic]
19
20        # Split synthetic data into training and validation sets
21        x_synth_train, x_synth_val, y_synth_train, y_synth_val =
    train_test_split(
22            x_synthetic, y_synthetic_one_hot,
    test_size=validation_split_ratio)
23
24        print(f"Training model on synthetic dataset {dataset_idx +
    1} for T = {10*num_samples}")
25
26        # Build the CNN model
27        model = cnn_model()
28
29        # Add callbacks for early stopping and model checkpointing
30        early_stopping = EarlyStopping(monitor='val_accuracy',
    patience=3, restore_best_weights=True)
31        model_checkpoint = ModelCheckpoint(
32            f'best_model_T={10*num_samples}_dataset_{dataset_idx +
    1}.keras',
33            save_best_only=True, monitor='val_accuracy', mode='max')
34
35        # Train the model with train/validation split
36        model.fit(
37            x_synth_train, y_synth_train,
38            validation_data=(x_synth_val, y_synth_val),
39            epochs=epochs,
40            verbose=1,
41            callbacks=[early_stopping, model_checkpoint]  # Include
    callbacks here
42        )
43
44        # Evaluate on original MNIST test dataset
45        test_loss, test_acc = model.evaluate(x_test, y_test_one_hot,
    verbose=0)
46        test_accuracies_per_T.append(test_acc)
47
48     avg_acc = np.mean(test_accuracies_per_T)
```

```
49     print(f"Average test accuracy for T = {10*num_samples}:
       {avg_acc:.4f}")
50     average_accuracies.append(avg_acc)
```

```
Displaying original images from MNIST dataset
```


```
Generating synthetic datasets for T = 2500
Displaying synthetic images for T = 2500, dataset 1
```




```
Generating synthetic datasets for T = 5000
Displaying synthetic images for T = 5000, dataset 1
```

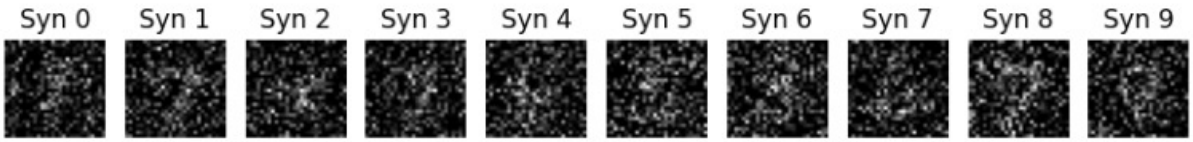


Figure B.3: Some samples from MNIST dataset and Synthetic datasets with varying T

Appendix C

GitHub Repository

The complete code and additional resources can be found in the GitHub repository:

https://github.com/Sabir-Mahmud/DPMix

Generated using Undegraduate Thesis LaTeX Template, Version 1.0. Department of Electrical and Electronic Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

This thesis was generated on Saturday 15th March, 2025 at 5:57pm.

B.Sc.

Engg.

EEE

BUET

Implementation and Performance Analysis of DPMix Algorithm

Sabir Mahmud

March

2025