SECURITY AND PRIVACY FOR SMART CITIES

IUDX Intern Documentation
for work on Differential Privacy
with Prof. K. Gopinath

SECURITY AND PRIVACY FOR SMART CITIES

PRESENTATION SLIDES FROM INTERNS WORKING WITH PROF. GOPINATH

During the course of the **sub-project (5) - Differential Privacy for Data Anonymization in Smart Cities**, Interns and others from the IUDX team worked extensively with Prof. Gopinath on collaborative and exploratory research.

The following Slides and Documents created by the IUDX team interns during the collaboration with Prof. Gopinath are included in this document**.**

1. Report on Privacy preserving Machine Learning – Pradeep Alapati
2. Privacy mechanisms in different countries – Pradeep Alapati
3. Improving InstaHide and Adding a Flavour of Randomized Response – Prajjwal Gupta & Pradeep Alapati
4. Applying Randomized Response mechanisms to Images and Studying its Effect on Downstream Processes – Prajjwal Gupta & Pradeep Alapati
5. Using Variational Autoencoders as PETs – Prajjwal Gupta & Pradeep Alapati
6. Privacy preserving Image Generation using Autoencoders with Quantized Latent Space - Prajjwal Gupta & Pradeep Alapati
7. PII – Abhin B
8. Knowledge Distillation – Abhin B
9. Knowledge Distillation tests – Cats vs Dogs – Abhin B

A Report on

Privacy Preserving Machine learning

(IUDX program unit)

Table Of Contents

What is Differential Privacy ?

Differential privacy is a concept in data privacy that aims to protect the privacy of individuals while allowing for the analysis and utilization of their data. It involves adding controlled noise or randomness to data before sharing it, in order to prevent the identification of specific individuals or sensitive information.

The basic idea of differential privacy is to provide statistical guarantees that the presence or absence of an individual's data in a dataset will not significantly impact the results of any queries or analyses performed on that dataset. This means that even if an attacker has access to the released data, they should not be able to determine with high certainty whether a particular individual's data is included in the dataset or not.

Differential privacy is typically achieved by adding carefully calibrated noise to the data during the data aggregation process, such as when computing statistical aggregates like counts, sums, or averages. This noise helps to mask the true values of individual data points, while still allowing for meaningful aggregate analyses to be performed.

Differential privacy has become an important concept in the field of data privacy, particularly in the context of data sharing for research, analysis, and policy-making purposes. It provides a rigorous and mathematically sound approach to protecting

individuals' privacy in data-driven applications, while still enabling valuable insights to be drawn from the data.

Mathematical Definition :

Let's assume we have two datasets, D and D', that differ in only one data point. In other words, D' can be obtained from D by adding or removing a single data point. These datasets are said to be "neighboring datasets".

A randomized algorithm A is said to satisfy epsilon-differential privacy if, for any pair of neighboring datasets D and D':

$P(A(D) \in S) \leq e^{\varepsilon} * P(A(D') \in S) + \boldsymbol{\delta}$

where:

- P(A(D) ∈ S) is the probability that the randomized algorithm A outputs a result in the set S when given dataset D.
- P(A(D') ∈ S) is the probability that the randomized algorithm A outputs a result in the set S when given dataset D'.
- epsilon($\boldsymbol{\varepsilon}$) is a privacy parameter that controls the strength of privacy guarantees. Smaller epsilon values provide stronger privacy guarantees.
- delta($\boldsymbol{\delta}$) is a small constant that represents an additional privacy parameter called "delta-differential privacy" that allows for a small amount of probability mass to be distributed

arbitrarily. It is typically set to a small value, such as 1e-5, to ensure that the privacy guarantee holds with high probability.

In essence, privacy is achieved by adding noise to the results thereby compromising on the accuracy which in return provides plausible deniability for a particular entity for which we are trying to protect the privacy of.

Plausible deniability:

- Flip a coin (the coin's bias is the probability that its outcome is head and it will be denoted as *p_head*.).
- If heads, return the answer in the entry.
- If tails, then flip a second coin and return "yes" if heads and "no" if tails.

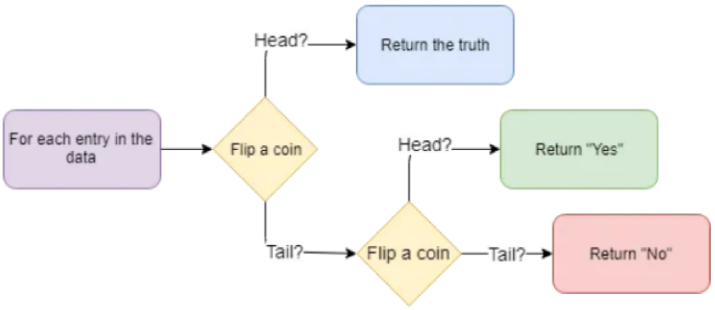


Figure 3. Flow diagram of the Differential privacy algorithm.

Now, each person is protected with "plausible deniability", because a person is plausible to deny the answer by the randomness of flipping a coin.

Some of the common noise mechanisms are laplacian and gaussian mechanisms.

Laplacian mechanism proposed by Cynthia Dwork has an upside of ensuring (ε, δ=0) differential privacy.

We will be using a gaussian mechanism for the work related to DP-ML.

For a simple and fun overview of DP :

(https://desfontain.es/privacy/friendly-intro-to-differential-privacy.html)

What does DP guarantee ?

- Privacy Protection: Differential privacy guarantees that the presence or absence of any individual's data in a dataset will not significantly impact the privacy of that individual. It prevents re-identification of specific individuals in the dataset, protecting their privacy and confidentiality.

- Robustness to Linkage Attacks: Differential privacy helps protect against linkage attacks, where an attacker combines the released data with external information to identify individuals. By adding controlled noise to the data, differential privacy makes it harder for an attacker to deduce sensitive information through linking with other data sources.

- Quantifiable Privacy: Differential privacy provides a quantitative measure of privacy through the privacy parameter epsilon. Smaller epsilon values indicate stronger privacy guarantees, allowing for fine-grained control over the trade-off between privacy and data utility.

- Flexibility: Differential privacy is a flexible concept that can be applied to various types of data and analysis tasks, including aggregate queries, machine learning algorithms, and other data analysis techniques. It can be used in different data sharing scenarios, such as centralized, distributed, and federated settings.

- Probabilistic Bound: Differential privacy provides a probabilistic bound on the privacy guarantee, allowing for a rigorous mathematical foundation. The delta parameter controls the probability of any additional privacy loss beyond epsilon, providing an additional level of privacy assurance.

- Reproducibility: Differential privacy allows for reproducible results, as the same input dataset will produce the same output with high probability, even with the added noise. This

ensures that analyses and results can be replicated, verified, and audited.

Overall, differential privacy provides a strong and quantifiable privacy guarantee that enables the sharing and analysis of data while protecting the privacy of individuals. It has become an important tool in the field of data privacy, with applications in various domains such as healthcare, finance, social sciences, and more.

To learn more on DP :

(Calibrating Noise to Sensitivity in Private Data Analysis)
https://journalprivacyconfidentiality.org/index.php/jpc/article/view/405

Extension of the concept of Differential privacy to Machine learning models :

Consider a case where we would like to publish a machine learning model which was trained on a sensitive/private dataset which we do not want to disclose, but we want to publish a model that makes some predictions based on that dataset.

People have discovered adversarial attacks that violate exactly this requirement.

Differential privacy can provide protection against several types of adversarial attacks in machine learning models, including:

- Membership Inference Attacks: Membership inference attacks involve an adversary trying to determine if a particular data point was used in the training data of a machine learning model. By exploiting the model's outputs, an adversary may be able to infer whether a data point was part of the training data, thereby violating privacy. Differential privacy can protect against membership inference attacks by adding controlled noise to the model's output, making it difficult for the adversary to determine if a specific data point was used in the training data or not.

  To know more : Shokri, R., Stronati, M., Song, C., & Shmatikov, V. (2017, May). Membership inference attacks against machine learning models. In 2017 IEEE symposium on security and privacy (SP) (pp. 3-18). IEEE.

- Reconstruction Attacks: Reconstruction attacks involve an adversary trying to reconstruct the sensitive data points used in the training data of a machine learning model based on the model's outputs. This can be done by exploiting the outputs of the model, such as the predicted probabilities or model parameters, to reconstruct the original data points.

Differential privacy can thwart reconstruction attacks by adding noise to the model's outputs, making it challenging for the adversary to accurately reconstruct the original data points.

To know more : Salem, A. M. G., Bhattacharyya, A., Backes, M., Fritz, M., & Zhang, Y. (2020). Updates-leak: Data set inference and reconstruction attacks in online learning. In 29th USENIX Security Symposium (pp. 1291-1308). USENIX.

- Model Poisoning Attacks: Model poisoning attacks involve an adversary trying to manipulate the training data to inject malicious data points with the aim of compromising the integrity or performance of the trained model. Differential privacy can provide protection against model poisoning attacks by perturbing the training data with controlled noise, making it harder for the adversary to maliciously manipulate the data points to poison the model.

  To know more : https://towardsdatascience.com/poisoning-attacks-on-machine-learning-1ff247c254db

- Query-based Attacks: Query-based attacks involve an adversary making repeated queries to a machine learning model to gain insight into the training data or infer sensitive

information about the individuals in the training data. Differential privacy can protect against query-based attacks by adding noise to the model's outputs or aggregating the query results in a privacy-preserving manner, preventing the adversary from extracting sensitive information through repeated queries.

To know more : Ilyas, A., Engstrom, L., Athalye, A., & Lin, J. (2018, July). Black-box adversarial attacks with limited queries and information. In International conference on machine learning (pp. 2137-2146). PMLR.

By incorporating differential privacy into machine learning models, organizations can mitigate the risk of these and other adversarial attacks, ensuring that the privacy of the individuals whose data is used for model training or inference is protected. However, it is important to note that the level of protection provided by differential privacy depends on the specific design and implementation of the differential privacy mechanism, and careful consideration should be given to the choice of privacy parameters and noise levels to achieve the desired privacy-utility trade-off.

In order to achieve differential privacy on Machine learning/ Deep learning models we have used the algorithm proposed by Abadi et.al.

While training the ML model, in the backpropagating step when the gradients are computed, the gradients are clipped to a particular threshold value and then the noise is added based on the required nose distribution (here the authors have picked gaussian distribution).

Algorithm 1 Differentially private SGD (Outline)

Input: Examples $\{x_1, \ldots, x_N\}$, loss function $\mathcal{L}(\theta) = \frac{1}{N}\sum_i \mathcal{L}(\theta, x_i)$. Parameters: learning rate η_t, noise scale σ, group size L, gradient norm bound C.
Initialize θ_0 randomly
for $t \in [T]$ **do**
  Take a random sample L_t with sampling probability L/N
  Compute gradient
  For each $i \in L_t$, compute $\mathbf{g}_t(x_i) \leftarrow \nabla_{\theta_t}\mathcal{L}(\theta_t, x_i)$
  Clip gradient
  $\bar{\mathbf{g}}_t(x_i) \leftarrow \mathbf{g}_t(x_i)/\max\left(1, \frac{\|\mathbf{g}_t(x_i)\|_2}{C}\right)$
  Add noise
  $\tilde{\mathbf{g}}_t \leftarrow \frac{1}{L}\left(\sum_i \bar{\mathbf{g}}_t(x_i) + \mathcal{N}(0, \sigma^2 C^2 \mathbf{I})\right)$
  Descent
  $\theta_{t+1} \leftarrow \theta_t - \eta_t \tilde{\mathbf{g}}_t$
Output θ_T and compute the overall privacy cost (ε, δ) using a privacy accounting method.

To learn more on differential privacy for machine learning/deep learning :

(Deep learning with Differential Privacy)(Abadi et.al)

(https://arxiv.org/abs/1607.00133)

Experimental Setup for replication of Concept and Results mentioned in Abadi et.al :

The authors have trained 2 models :

a) Without DP (baseline)
b) With DP

in order to quantify the privacy results and the impact of epsilon on accuracy.

There is no mention of the scaling mechanism for the input parameters in the paper. So in order to test its impact we have performed Min-Max and Standard Scaling before and after PCA (as mentioned in the paper)

The authors use a Differentially Private PCA, but due to time constraints we use a generic PCA.

In addition to this we have also tested the impact of momentum in the training process. We have tested with the momentum values of 0 and 0.9.

With the mechanism and parameters mentioned in the paper and the above mentioned scaling and momentum in mind, we have performed experiments using 2 different Frameworks :
i) Pytorch and ii) tensorflow.

On top of these frameworks, in order to apply differential privacy, we have used libraries such as: i) Opacus (for DP on pytorch) and ii) TF-privacy (for DP on tensorflow) .

Better results and faster training is achieved using Opacus and hence Pytorch-Opacus combo is our choice for future experiments.

The machine learning model coded for the experiment is basically a hand-written digit classifier for which the dataset it is trained on is the famous MNIST dataset.

(MNIST : 70,000 28x28 black-and-white images of handwritten digits (0-9) extracted from two NIST databases)

To briefly categorize the experiments performed :

1) Baseline model in
   a) pytorch (linear layers)
   b) tensorflow (linear layers)
2) Baseline model integrated with a DP-SGD privacy engine in
   a) pytorch-opacus (linear, batchnorm and CNN layers)
   b) tensorflow-tf privacy (linear layers)

One important issue that had to be addressed was the incompatibility of batch normalization for differential privacy.

(https://arxiv.org/pdf/2006.10919.pdf)

Code implemented for the experiments :

https://github.com/datakaveri/differential-privacy-ml

(accessible only by members of IUDX-datakaveri repository)

The main difference between 1) and 2) is that one will find once gone through the code in the github repo is the integration of the privacy engine.

Example code snippet :

```
privacy_engine = PrivacyEngine()
model, optimizer, train_loader = privacy_engine.make_private_with_epsilon(
    module=model,
    optimizer=optimizer,
    data_loader=train_loader,
    epochs=epochs,
    target_epsilon=epsilon,
    target_delta=delta,
    max_grad_norm=max_grad_norm
)
```

To install opacus on our runtime :

```
!pip install opacus
```

To import the privacy engine that we require from opacus and the module validator that check for parts of code that are incompatible with opacus (ie DP incompatible) :

```
from opacus import PrivacyEngine
from opacus.validators import ModuleValidator
```

To track and print the epsilon throughout the epochs :

```
epsilon = privacy_engine.get_epsilon(delta)
print('Trained Epoch: {} with loss= {:.6f}, accuracy= {:.2f}, and epsilon= {:.2f}'.format(
        epoch,
        epoch_loss / count,
        100. * correct / len(train_loader.dataset),
        epsilon
))
```

To check for incompatibility and change the type of layers (for example batch norm in this case) :

```
model = Net()
model = model.to(device)
errors = ModuleValidator.validate(model, strict=False)
print("ERRORS: ", errors[-5:])
model = ModuleValidator.fix(model)
errors = ModuleValidator.validate(model, strict=False)
print("NEW ERRORS: ", errors[-5:])
optimizer = optim.SGD(model.parameters(), lr=lr, momentum=momentum)
```

All these above code snippets can be better understood when we go through the below documentation of Opacus and the code in the github front the beginning.

Since moving forward Opacus will be preferred, here is the documentation for its usage :

(https://opacus.ai/api/)

Results :

Here are the results of the experiments performed :

	Sigma = 1 gradclipnorm = 4 epochs = 100 learningrate = 0.05 delta = 1e-5	No Scaling	Min-Max Scaling before PCA	Min-Max scaling after PCA	Standard Scaling before PCA	Standard Scaling after PCA
Momentum = 0.9	**Pytorch baseline**	97.87	97.56	97.4	97.51	97.53
	Pytorch DP	93.73	96.45	92.17	94.68	95.19
	TF baseline	-	-	-	-	-
	TF-Privacy	-	-	-	-	-
Momentum = 0	**Pytorch baseline**	97.62	97.76	97.61	97.58	97.52
	Pytorch DP	94.6	95.82	90.56	95.62	95.79
	TF baseline	98.21	98.09	98.16	98.26	98.01
	TF-Privacy	96.14	96.01	90.39	95.76	95.89
	Average Accuracy	96.36166667	96.94833333	94.38166667	96.56833333	96.655

DPML with Opacus										
Sigma	No Scaling		MinMax Before		MinMax After		Standard Before		Standard After	
	m=0	m=0.9	m=0	m=0.9	m=0	m=0.9	m=0	m=0.9	m=0	m=0.9
3.86718	89.8	89.29	94.47	89.25	89.74	87.14	93.61	89.47	93.72	87.14
2.13867	91.56	90.84	95.65	92.41	90.18	90.18	95.18	91.17	95.07	90.18
1.12548	93.87	92.9	95.72	95.23	90.42	93.07	95.79	93.04	95.5	93.07
0.8023	94.34	93.59	95.94	96.31	90.43	95.07	95.68	94.84	95.68	95.07
0.6269	94.97	94.24	96.05	97.03	90.56	96.18	95.79	95.78	95.74	96.18

Here are some research questions and its answers we are able to infer from our experiments :

Question	Resolution
What is the relationship between momentum and accuracy in DP-SGDM?	For differentially private model training, momentum shows an increasing accuracy trend but is outperformed by a model that is trained without momentum
Momentum should intuitively affect the rate of convergence of the model, so why does it have an affect on the overall accuracy of the model? Is the 'saddle hypothesis' worth exploring?	Not worth exploring
Can we safely assume the usage of momentum in Abadi et al. and can we take the results obtained in their paper at face value?	In Abadi: momentum is not used Tensor Flow is used with a custom implementation of DP
Is our implementation of DP-SGD directly comparable to Abadi et al.?	Yes, in both TensorFlow and Pytorch models. We can progress with Pytorch as it is comparable to tensorflow with caveats noted in D2

16

Does setting sigma (noise variance) to 0 give the same results as non-DP trained models?	Not possible to set sigma to 0 in DP models
What is the relationship between scaling and PCA in machine learning? Is there a particular scaling method that works better for PCA?	Scaling should be done before PCA, scaling method depends on application

Work done alongside Prof.Gopinath:

We started collaborating with Prof.Gopinath in order to come up with "cheaper" methods for privacy on AV/image data/ML, with better [cryptographically] grounded privacy guarantees and also to address the question of “Whose privacy is being preserved in PPML on image datasets? ”

InstaHide is a powerful approach which allows deep learning on encoded images with certain cryptographic guarantees.

We wanted to improve these guarantees and address and solve any drawbacks associated with this approach.(Slides for reference)

Taking inspiration from the paper- Instahide, we conducted basic experiments in order to examine RR mechanisms for Images and studying its effect on downstream processes.

Pixel Randomization Techniques Used

For a given block of K X K pixels, we first create a mask using one of the techniques below, then add the mask to the block followed by taking the absolute value and clipping to the range 0-255.

Pixel Flipping	Pixel Randomization	Laplacian Noise Addition
Since pixel values range from 0-255, for a pixel value i, it's flipped pixel is the absolute value |i - 255|. The mask is a matrix with all values as -255.	Pixel values are randomized by sampling a random number between -255 and 255, adding it to the pixel, taking its absolute value and clipping it's value to 0-255. The pixel values are directly not replaced by a random value because we are following the concept of application of a mask	A random sample from a laplacian distribution with location 0 and scale 255/epsilon is used to create the mask.

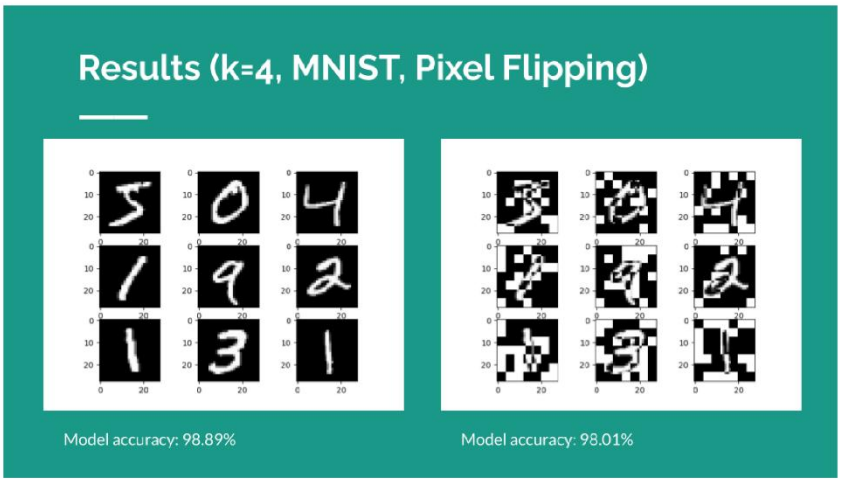
Here is the basic overview of the code for this:

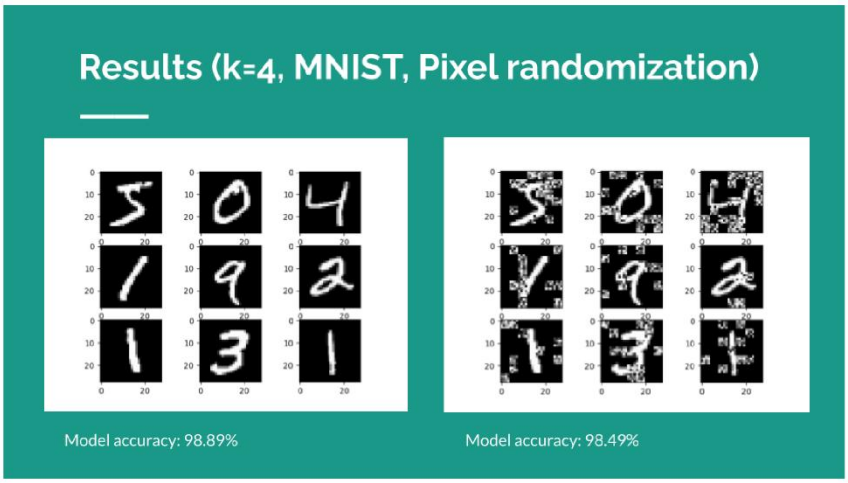
```
def perturb(self, block) -> Any:
        """
        Perturbs a block of K X K pixels
        """
        mask = None
        if self.mode == "flip":
            mask = -255

        elif self.mode == "random":
            mask = np.random.randint(-255, 255, size=(self.K, self.K))

        elif self.mode == "dp":
            with open('config.json') as config_file:
                configs = json.load(config_file)
            epsilon = configs["epsilon_dp_flip"]
            mask = np.random.laplace(loc=0, scale=255/epsilon, size=(self.K, self.K))

        perturbation = block + mask
        perturbation = np.abs(perturbation)
        perturbation[perturbation > 255] = 255
        perturbation[perturbation < 0] = 0
        return perturbation
```

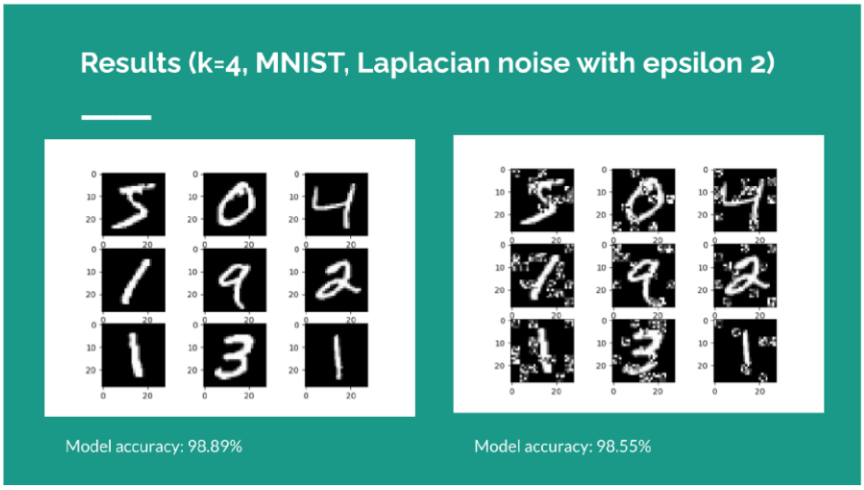
18

We ran these experiments on the standard MNIST 28x28 pixel dataset.

Here are a few results for this experiment:







In addition to this, we have conducted experiments with overlap in the K x K block such that it iterates by a single pixel and also added gaussian noise with specified probability/coin flip.

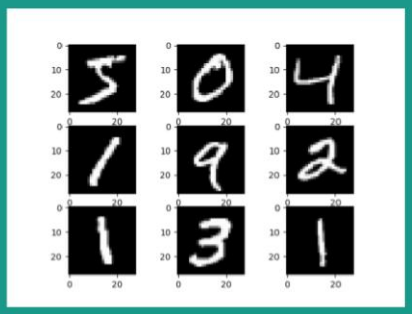
Here are some basic code snippets for this experiment:

```
widthItr = (image.shape[1] )
heightItr = (image.shape[0] )

for i in range(widthItr-self.K):
    for j in range(heightItr-self.K):
        # TODO: Take care of edge cases
        block = image[j:j+self.K, i:i+self.K]
        do_i_perturb = np.random.rand()
        if do_i_perturb < configs["flip_prob1"]:
            do_i_really_perturb = np.random.rand()
            if do_i_really_perturb < configs["flip_prob2"]:
                image[j:j+self.K, i:i+self.K] = self.perturb(block)
            else:

                pass
        else:

            pass
  return image
```

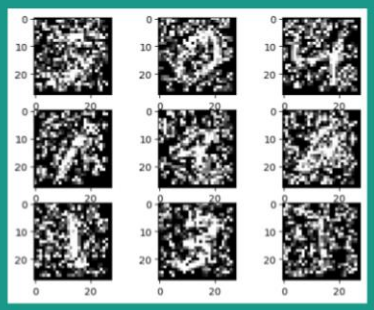
```
for i in range(widthItr):
    for j in range(heightItr):
        # TODO: Take care of edge cases
        block = image[j*self.K:(j+1)*self.K, i*self.K:(i+1)*self.K]
        do_i_perturb = np.random.rand()
        if do_i_perturb < configs["flip_prob1"]:
            do_i_really_perturb = np.random.rand()
            if do_i_really_perturb < configs["flip_prob2"]:
                image[j*self.K:(j+1)*self.K, i*self.K:(i+1)*self.K] = self.perturb(block)
            else:
                image[j*self.K:(j+1)*self.K, i*self.K:(i+1)*self.K]=gaussian_filter(block, sigma=1)


        else:
            image[j*self.K:(j+1)*self.K, i*self.K:(i+1)*self.K]=gaussian_filter(block, sigma=1)

return image
```

Here are a few results for this experiment:

Results (k=2, MNIST, Laplacian noise with epsilon 2, overlap)

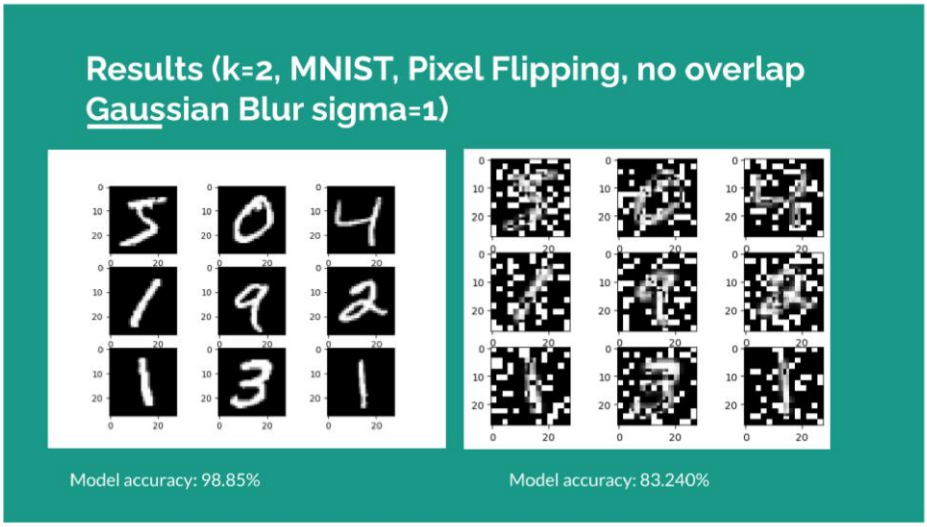


Model accuracy: 98.85%



Model accuracy: 95.720%

21



Here is the link for code implemented to run experiments on randomized response:
https://github.com/datakaveri/PPML-Research.git

Research using Autoencoders / Variational Autoencoders

In order to come up with a better approach that quantifies the privacy loss while obfuscation before downstream processing, we are currently looking at approaches with autoencoders/variational autoencoders.

An autoencoder is a type of artificial neural network that is used for unsupervised learning, specifically for dimensionality reduction or

feature learning. It is designed to encode input data into a lower-dimensional representation and then decode it back to its original form.

The basic structure of an autoencoder consists of an encoder and a decoder. The encoder takes the input data and maps it to a lower-dimensional latent space representation, also known as the bottleneck layer or code. The decoder then takes this code and reconstructs the original input data from it. The objective of the autoencoder is to minimize the reconstruction error, i.e., make the output as similar as possible to the input.

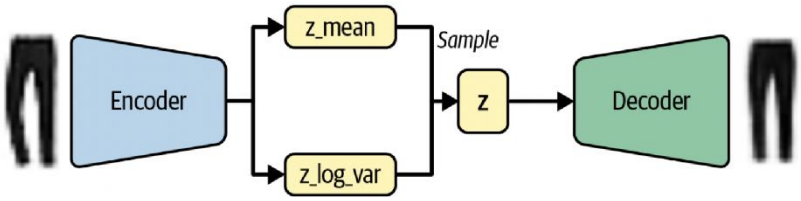
Autoencoders can be used for various purposes, such as data compression, denoising, and anomaly detection. By learning a compressed representation of the data, autoencoders can capture important features and discard irrelevant or noisy information.

Variational Autoencoder (VAE) is an extension of the autoencoder that introduces probabilistic modeling to the latent space. Unlike a traditional autoencoder, which maps inputs deterministically to a fixed code, a VAE models the latent space as a probability distribution. This enables generating new data samples by sampling from the learned distribution.

In a VAE, the encoder maps the input data to the parameters of a probability distribution in the latent space, typically a Gaussian distribution. The decoder then takes samples from this distribution and reconstructs the input data. During training, the VAE learns to

optimize the reconstruction error as well as maximize the likelihood of the latent space distribution.

By introducing a probabilistic component, VAEs allow for smooth interpolation between data points in the latent space, enabling meaningful generation of new samples. This makes VAEs useful for tasks such as image generation, data synthesis, and unsupervised clustering.



In summary, autoencoders and variational autoencoders are neural network architectures used for unsupervised learning. Autoencoders learn a compressed representation of the input data, while VAEs introduce probabilistic modeling to enable data generation and interpolation in the latent space.

We intend to bring in the concepts of privacy in Autoencoders by quantizing the encoding to the nearest block in the quantized space.This method does take a toll on the utility but intuitively offers some privacy with the loss in accuracy.

In a nutshell what is expected to happens is, we pass the private dataset through the private-autoencoder and then use the

generated dataset from the private-autoencoder for further downstream processing.

Currently we are researching to quantify the privacy gained and linking it to the accuracy in a similar manner to that for DP-SGD.

Here is the link for all slides used for discussion in meeting with Prof. Gopinath : DPML Research Resources with Prof Gopinath

---

References and Resources :

1. Dwork, C., McSherry, F., Nissim, K., & Smith, A. (2016). Calibrating noise to sensitivity in private data analysis. Journal of Privacy and Confidentiality, 7(3), 17-51.
2. Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., & Zhang, L. (2016, October). Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security (pp. 308-318).
3. https://github.com/datakaveri/differential-privacy-ml (Github repo of experiments performed)
4. https://desfontain.es/privacy/friendly-intro-to-differential-privacy.html (Simple and interactive explanation of DP)
5. https://discuss.pytorch.org/t/why-batchnorm-layer-is-not-compatible-with-dp-sgd/154208 (Effect of Batch Normalization in DP-SGD)

6. https://opacus.ai/api/ (Opacus documentation)
7. Shokri, R., Stronati, M., Song, C., & Shmatikov, V. (2017, May). Membership inference attacks against machine learning models. In 2017 IEEE symposium on security and privacy (SP) (pp. 3-18). IEEE.
8. Salem, A. M. G., Bhattacharyya, A., Backes, M., Fritz, M., & Zhang, Y. (2020). Updates-leak: Data set inference and reconstruction attacks in online learning. In 29th USENIX Security Symposium (pp. 1291-1308). USENIX.
9. https://towardsdatascience.com/poisoning-attacks-on-machine-learning-1ff247c254db
10. Ilyas, A., Engstrom, L., Athalye, A., & Lin, J. (2018, July). Black-box adversarial attacks with limited queries and information. In International conference on machine learning (pp. 2137-2146). PMLR.
11. Huang, Y., Song, Z., Li, K., & Arora, S. (2020, November). Instahide: Instance-hiding schemes for private distributed learning. In International conference on machine learning (pp. 4507-4518). PMLR.
12. Raynal, M., Achanta, R., & Humbert, M. (2020). Image obfuscation for privacy-preserving machine learning. arXiv preprint arXiv:2010.10139.
13. Chen, Q., Xiang, C., Xue, M., Li, B., Borisov, N., Kaarfar, D., & Zhu, H. (2018). Differentially private data generative models. arXiv preprint arXiv:1812.02274.

14. Jiang, D., Zhang, G., Karami, M., Chen, X., Shao, Y., & Yu, Y. (2022). DP $^ 2$-VAE: Differentially Private Pre-trained Variational Autoencoders. arXiv preprint arXiv:2208.03409.
15. Dockhorn, T., Cao, T., Vahdat, A., & Kreis, K. (2022). Differentially private diffusion models. arXiv preprint arXiv:2210.09929.

Application of Privacy mechanisms by various countries in cases relevant to public release.

Pradeep Alapati

29th June, 2023

Introduction

In today's digital age, privacy concerns have become increasingly important as individuals and organizations generate vast amounts of data. To address these concerns, countries around the world are implementing various privacy techniques, including Differential Privacy (DP), to protect sensitive information while enabling the release of useful data for public use.

This report aims to provide an overview of the current state of application of DP and other privacy techniques by different countries in cases relevant to public release.

Differential Privacy (DP) :

Differential Privacy is a framework that aims to balance data privacy and data utility by adding controlled noise to statistical queries or data releases. It provides mathematical guarantees that the privacy of individuals in a dataset is preserved, even when the data is analyzed or released publicly. DP has gained significant attention and has been adopted by several countries in various applications.

Implementations by countries

Countries generally employ various techniques and methodologies to release statistics in a privacy-preserving manner. Here are some commonly used approaches:

Aggregation and Suppression: One common technique is to aggregate data at a higher level, such as regional or national, to prevent the identification of individual respondents. Additionally, suppression is applied to sensitive cells or small cell counts to ensure that specific information cannot be deduced from the published statistics. This method helps protect the privacy of individuals or businesses contributing to the data while still providing useful aggregated information.

Noise Injection: Another approach is to add random noise or perturbations to the data before publication. This technique ensures that individual values are altered, making it difficult to identify specific respondents. Differential privacy is a popular framework that quantifies the amount of noise added to achieve a certain level of privacy protection while preserving data utility.

Data Masking and Swapping: Countries may apply data masking techniques to de-identify or anonymize sensitive information. This involves replacing identifiable values with randomized or synthesized data that retain the statistical properties of the original dataset. Another method is

data swapping, where values are exchanged between different records, preserving statistical characteristics while obfuscating the relationship between the data and the individuals or entities.

Secure Data Enclaves: Some countries establish secure data enclaves or research centers where authorized researchers can access and analyze sensitive microdata under strict privacy and confidentiality protocols. These environments ensure that individual-level data remains protected while enabling advanced analysis and research.

Access Control and Encryption: Robust access control mechanisms and encryption techniques are employed to secure statistical databases and restrict unauthorized access. Strong authentication, encryption of data at rest and in transit, and data access logs are used to maintain data privacy and integrity.

Data Sharing Agreements: Countries may establish data sharing agreements with trusted organizations or researchers, ensuring strict confidentiality and privacy safeguards. These agreements outline the conditions for data use, access, and storage, ensuring compliance with privacy regulations.

Privacy Impact Assessments: Prior to releasing statistics, countries often conduct privacy impact assessments to evaluate the potential privacy risks and determine appropriate safeguards. This process helps identify and mitigate privacy concerns before the data is made public.

1) **United States** :

a) The United States has been a pioneer in implementing DP in public release cases. The U.S. Census Bureau adopted DP for the 2020 decennial census to ensure individual privacy while providing accurate population statistics. By introducing noise into the data, the Bureau maintained privacy protections, making it challenging to identify individuals while still obtaining valuable statistical information about population counts, geography, and demographic characteristics.

   This is the first time that a country has released most of its subpopulation counts with formal privacy protections, although certainly not the first time that other official counts have been perturbed for the purpose of disclosure avoidance.

   https://privacytools.seas.harvard.edu/blog/differential-privacy-2020-census-how-can-we-make-data-both-private-and-useful

b) The USA's National Library of Medicine has proposed Privacy-preserving data sharing via probabilistic modeling.Their approach transforms the problem of designing the synthetic data into choosing a model for the data, allowing also the inclusion of prior knowledge, which

improves the quality of the synthetic data. They demonstrate empirically, in an epidemiological study, that statistical discoveries can be reliably reproduced from the synthetic data. They expect the method to have broad use in creating high-quality anonymized data twins of key datasets for research.

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8276015/

2) **Canada**:

Statistics Canada has strict privacy policy measures that have been developed over decades of collecting data and releasing official statistics. To address the new requirements for operating in the cloud, they consider a class of new cryptographic techniques called Privacy Preserving Technologies (PPTs) that might help increase utility by taking greater advantage of technologies such as the cloud or machine learning while continuing to preserve the agency's security posture.

For output level privacy, they are examining the use of Differential privacy and for input level privacy, they are examining homomorphic encryption and secure multiparty computation.

They are also exploring the use of neural networks in the context of privacy with Federated learning and Split learning.

https://www.statcan.gc.ca/en/data-science/network/privacy-preserving

3) **Australia**:

The Data Access and Confidentiality Methodology Unit (DACMU) of the Australian Bureau of Statistics aims to develop a new confidentiality method that balances the trade-off between data utility and privacy protection. This method is designed to enhance the usefulness of statistical collections while ensuring state-of-the-art privacy safeguards for data providers. The goal is to apply this method consistently across a wide range of household and business statistics publications. Currently, the ABS relies on passive confidentiality, where data providers must notify the ABS to protect their privacy, but this approach has limitations. To address these issues, they explore the implementation of Pufferfish differential privacy (DP) using log-Laplace multiplicative perturbation. If successful, this approach could be applied to various data sources and significantly improve the utility of the information.

The current confidentiality method used by the ABS, called consequential suppression, has become inadequate for several reasons. Firstly, it hampers the ABS's ability to meet the increasing demand for more detailed statistics, especially in complex scenarios involving multiple small geographic areas. Additionally, consequential suppression limits the ABS's flexibility to produce timely and user-specific outputs due to the need for further suppression when geospatial differencing is involved. Research indicates that suppression is less

effective than perturbation in terms of privacy protection. Moreover, consequential suppression fails to meet the user demand for detailed unit-level analysis using alternative data sources and requires substantial resources. As the ABS plans to rely more on administrative datasets, direct engagement with data providers through surveys will be reduced.

https://www.abs.gov.au/statistics/research/confidentiality-abs-business-data-using-pufferfish-differential-privacy

4) **Others**:

Countries such as France ,U.K, Netherlands and also the European Union are actively looking to adopt Differential privacy for releasing public statistics in contrast to the generic methods such as Aggregation,suppression,noise injection,data masking,swapping,etc.

References

- https://privacytools.seas.harvard.edu/blog/differential-privacy-2020-census-how-can-we-make-data-both-private-and-useful
- https://hdsr.mitpress.mit.edu/pub/gl9z7ehf/release/8
- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8276015/
- https://www.abs.gov.au/AUSSTATS/abs@.Nsf/Previousproducts/1504.0Main%20Features2Dec%202020?opendocument&tabname=Summary&prodno=1504.0&issue=Dec%202020&num=&view=
- https://www.abs.gov.au/statistics/research/confidentiality-abs-business-data-using-pufferfish-differential-privacy
- https://jameshbailie.github.io/Papers/papers/UNECE2019.pdf
- https://www.statcan.gc.ca/en/data-science/network/privacy-preserving
- https://arxiv.org/pdf/2102.08847.pdf
- https://arxiv.org/pdf/2304.02107.pdf
- https://www.cbs.nl/-/media/_pdf/2021/49/privacy-preserving-techniques-and-statistical-disclosure-control.pdf

Companies that offer Differential Privacy products or services

1. **Tamult labs** : https://www.tmlt.io/
2. **Sensor tower** : https://sensortower.com/blog/sensor-tower-introduces-differential-privacy
3. **Aircloak** : https://aircloak.com/solutions/how-it-works/
4. **Neptune.ai**: https://neptune.ai/blog/using-differential-privacy-to-build-secure-models-tools-methods-best-practices
5. **Privitar** : https://www.privitar.com/glossary/differential-privacy/#:~:text=Differential%20Privacy%20is%20a%20characteristic,data%20processed%20by%20that%20algorithm.

Improving InstaHide and adding a flavour of RR

Prajjwal Gupta
Pradeep Alapati

Problem Statement

InstaHide is a powerful approach which allows deep learning on encoded images with certain cryptographic guarantees.

We want to improve these guarantees and address and solve any drawbacks associated with this approach.

Approach

First we study the algo step by step and identify any potential weaknesses or limitations solving which we could possible achieve some improvement in the security/privacy aspect.

Next we look into some existing work on RR which could possibly be incorporated into the algo.

Solutions

We also look at possible solutions to the problems or limitations discovered in our analysis of the algorithm

The InstaHide Algorithm
Step 1: A secure mixup procedure

Issue 1: Comparison to DP-like algos

This step involves a mixup operation with k-1 images. The understanding is, the larger the value of k, the more is the security or obscurity of the image. Since we are in a way comparing the algo to DP-based approaches (in terms of utility), can epsilon and k be used as a basis of comparison?

In DP-based approaches, the privacy guarantee is measured by epsilon. So can we produce utility vs k and utility vs epsilon graph and make some inference such as for the same utility guarantee, what would be the corresponding epsilon for a DP-based approach?

Intuition is, if InstaHide offers more utility then a reasonable value of k would correspond to a high value of epsilon

Issue 2: Dependance on a public dataset

The dependance on a public dataset could pose a development or deployment issue. the authors use ImageNet which is ideal due to its large size. But most practical applications for example medical scans, would consist of images of very high resolution. In such a case where resolution of private set > resolution of public set, we might face a problem since large public datasets are not available in every resolution.

Solution: Can we use GAN-based techniques to sample patterns and assign labels to these patterns? Since we are working on obscuring patterns in the private dataset, this could work. The sampling procedure would have to be carefully designed.

Issue 3: Motivation for cross-dataset InstaHide over Inside-Dataset InstaHide

"Some privacy sensitive datasets, (e.g. CT or MRI scans), feature images with certain structure patterns with uniform backgrounds" - InstaHide pg4

In such a case of a special dataset, the presence uniform backgrounds can be taken as an a priori for the adversary (through side-channels/intuition). Then these areas are vulnerable as in, with lesser complexity, the adversary can decode the one-time keys from the public dataset which were used.

Issue 4:
Change in nature of the classification objective

This procedure converts the task from single label classification problem to multi-label classification problem. In such a case, in a practical scenario, how will the deployment and output processing and interpretation be handled?

"Either scheme above by default applies InstaHide during inference, by averaging predictions of multiple encryptions (e.g. 10) of a test sample" - InstaHide pg5

"One can also choose not to apply InstaHide during inference. We found in our experiments (Section 5) that it works for low-resolution image datasets such as CIFAR-10 but it does not work well with a high-resolution image dataset such as ImageNet." - InstaHide pg5

The InstaHide Algorithm
Step 1: A secure mixup procedure

Issue 4:
Does random sign flipping not scale well with increasing value of k?

"A priori. It may seem that using a different mask for each training sample would completely destroy the accuracy of the trained net, but as we will see later it has only a small effect when k is small. Mathematically, this seems reminiscent of the phase retrieval problem" - InstaHide pg4.

If k is small, then don't we lose out on the security and privacy aspects?

Towards a better scheme for replacing/improving random sign flipping

Randomized Response

We are interested in a randomized output of this step, we essentially want some fuzzy output which still retains some properties of the original data which can be exploited in any downstream processing.

DP is a widely accepted RR mechanism.

DP based mechanisms require us to define:

> Who's privacy are we protecting? (One pixel, multiple pixels, convolutions etc)

> How do we define neighbouring datasets? (Depending on the answer to previous question)

> Based on the definition of neighboring dataset, how do we calculate the sensitivity of the "query"?

Paper: Differential Privacy for Image Publication

by Liyue Fan

https://webpages.charlotte.edu/lfan4/pdf/TPDP2019.pdf

They extend the standard differential privacy notion to image data, which protects individuals, objects, and/or their features. They develop differentially private methods for two popular image obfuscation techniques, i.e., pixelization and Gaussian blur.

> Who's privacy are we protecting? (One pixel, multiple pixels, convolutions etc)

They work on protecting the participation of an object in an image which occupy at most m pixels.

Paper: Differential Privacy for Image Publication

by Liyue Fan

> How do we define neighbouring datasets? (Depending on the answer to previous question)

Definition 1. *[ϵ-Image Differential Privacy] A randomized mechanism $\mathcal{A}$ gives ϵ-differential privacy if for any neighboring images I_1 and I_2, and for any possible output $\overline{I} \in Range(\mathcal{A})$,*

$$Pr[\mathcal{A}(I_1) = \overline{I}] \leq e^{\epsilon} \times Pr[\mathcal{A}(I_2) = \overline{I}] \quad (1)$$

where the probability is taken over the randomness of $\mathcal{A}$.

Definition 2. *[m-Neighborhood] Two images I_1 and I_2 are neighboring images if they have the same dimension and they differ by at most m pixels.*

Paper: Differential Privacy for Image Publication

by Liyue Fan

> Based on the definition of neighboring dataset, how do we calculate the sensitivity of the "query"?
>
> The author proposed two algos: DP-Pix and DP-Blur:

`DP-Pix`. In a nutshell, the algorithm first performs pixelization on an input image, i.e., by computing the average pixel value for each grid cell, and applies perturbation to the pixelized image. Intuitively, the global sensitivity of pixelization for each b x b grid cell is $\frac{255m}{b^2}$. The corresponding Laplace mechanism [2] is adopted to draw random noises, in order to achieve ϵ-differential privacy.

`DP-Blur`. The initial design of the algorithm first applies Laplace perturbation to each pixel and then performs Gaussian blur, in order to produce a "smooth" output image. The global sensitivity of direct pixel-wise operation is $255m$, which induces larger noises compared to the pixelization based approach. To reduce the effect of the noise, we first run `DP-Pix` on the input image, using a small cell width b_0. We then upsample the private pixelization to the original size and perform Gaussian blur.

DP as a mechanism for RR on pixel values (images) is not well studied in the literature and seems to be an open problem. Researchers have proposed mechanisms for downstream processing (DP-SGD, PATE, Label-DP).

Answering the key questions required for a DP-mechanism setup, we can look into creating our own definition and solution which fits the requirements demanded/desired by InstaHide

Applying RR mechanisms to Images and studying its effect on downstream processes

Prajjwal Gupta
Pradeep Alapati

Pixel Randomization Techniques Used

For a given block of K X K pixels, we first create a mask using one of the techniques below, then add the mask to the block followed by taking the absolute value and clipping to the range 0-255.

Pixel Flipping

Since pixel values range from 0-255, for a pixel value i, it's flipped pixel is the absolute value |i - 255|.

The mask is a matrix with all values as -255.

Pixel Randomization

Pixel values are randomized by sampling a random number between -255 and 255, adding it to the pixel, taking its absolute value and clipping it's value to 0-255.

The pixel values are directly not replaced by a random value because we are following the concept of application of a mask

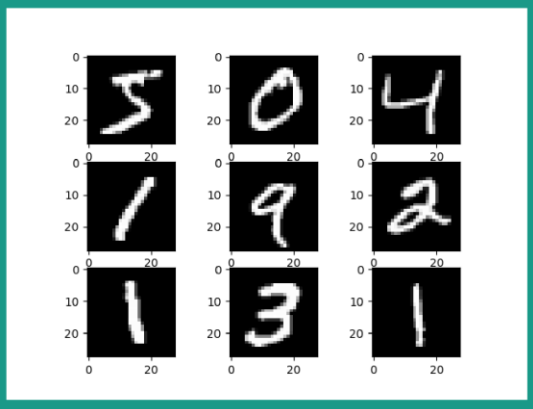
Laplacian Noise Addition

A random sample from a laplacian distribution with location 0 and scale 255/epsilon is used to create the mask.

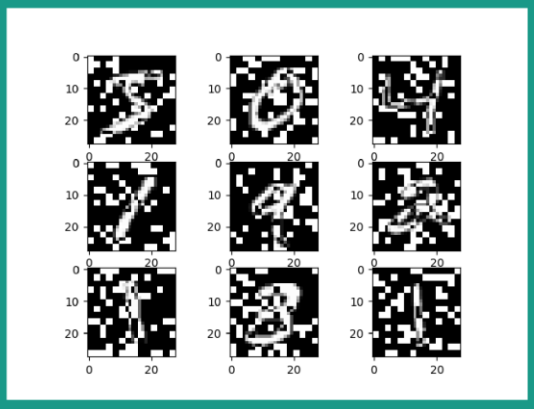
The code

```
def perturb(self, block) -> Any:
        """
        Perturbs a block of K X K pixels
        """
        mask = None
        if self.mode == "flip":
            mask = -255

        elif self.mode == "random":
            mask = np.random.randint(-255, 255, size=(self.K,
self.K))

        elif self.mode == "dp":
            with open('config.json') as config_file:
                configs = json.load(config_file)
            epsilon = configs["epsilon_dp_flip"]
            mask = np.random.laplace(loc=0, scale=255/epsilon,
size=(self.K, self.K))

        perturbation = block + mask
        perturbation = np.abs(perturbation)
        perturbation[perturbation > 255] = 255
        perturbation[perturbation < 0] = 0
        return perturbation
```

Results (k=2, MNIST, Pixel Flipping)

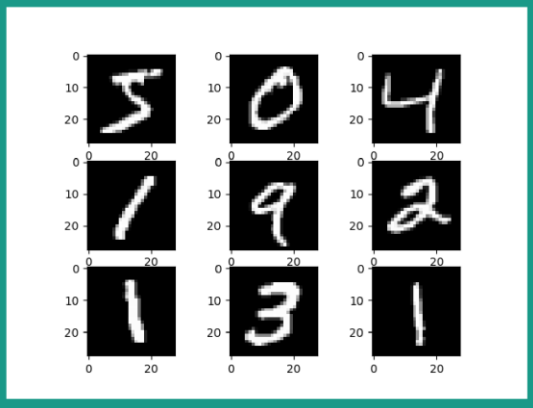


Model accuracy: 98.89%

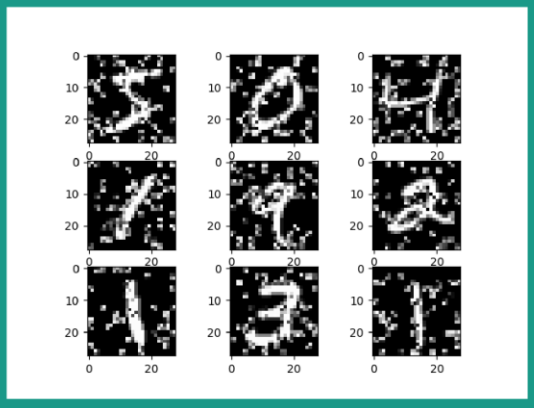


Model accuracy: 96.68%

Results (k=2, MNIST, Pixel randomization)

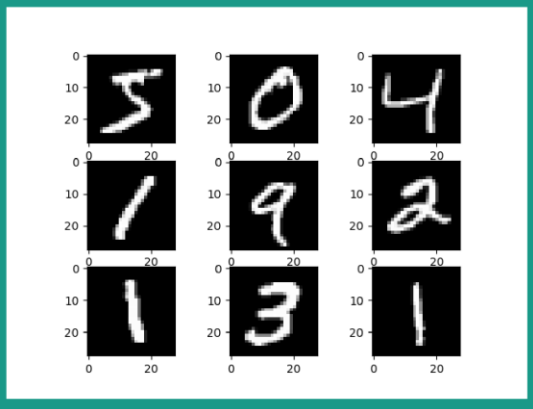


Model accuracy: 98.89%

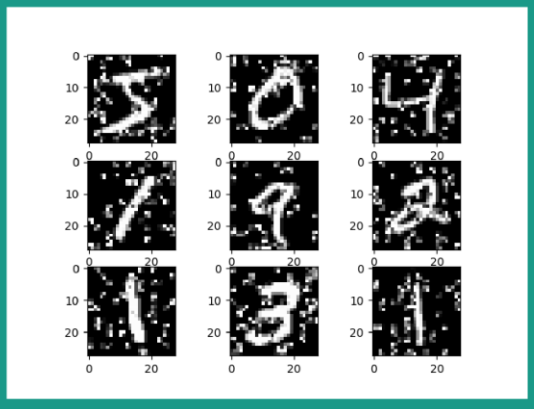


Model accuracy: 98.47%

Results (k=2, MNIST, Laplacian noise with epsilon 2)

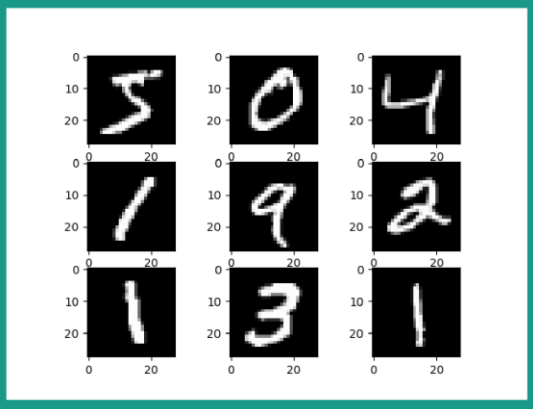


Model accuracy: 98.89%

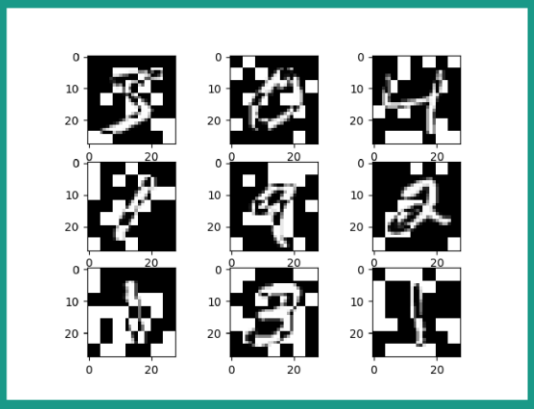


Model accuracy: 98.52%

Results (k=4, MNIST, Pixel Flipping)

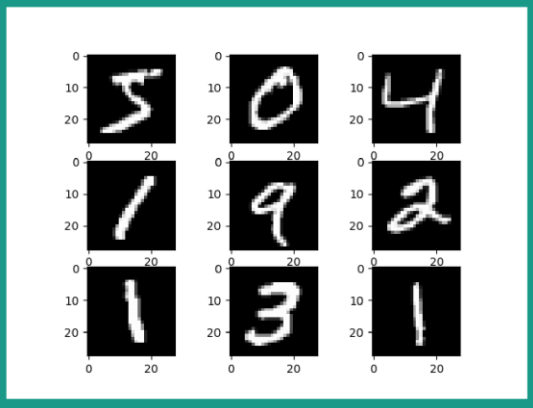


Model accuracy: 98.89%

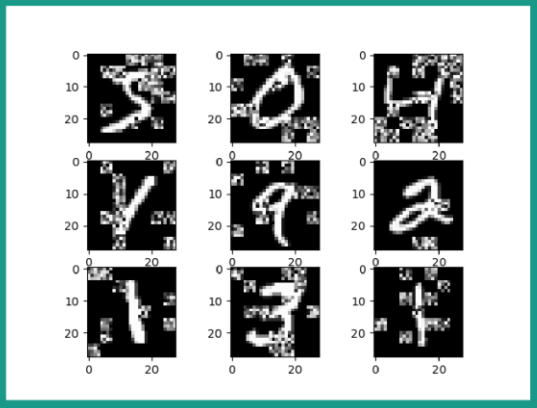


Model accuracy: 98.01%

Results (k=4, MNIST, Pixel randomization)

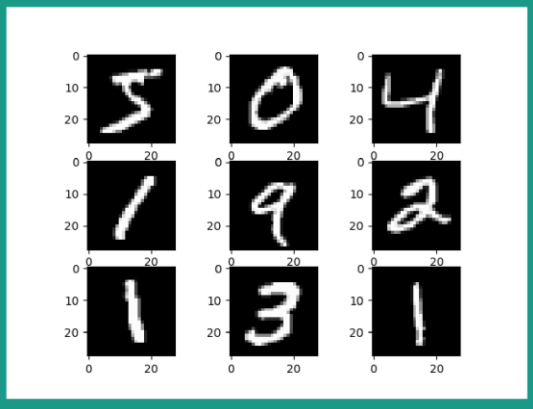


Model accuracy: 98.89%

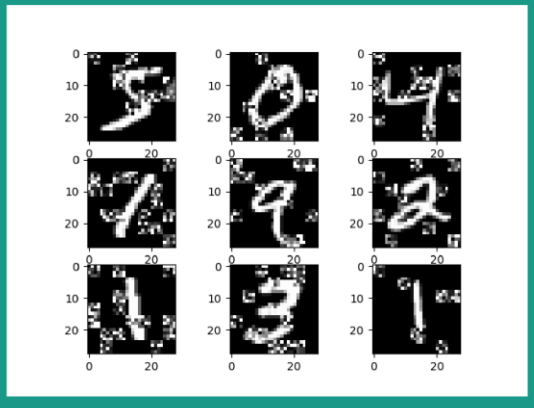


Model accuracy: 98.49%

Results (k=4, MNIST, Laplacian noise with epsilon 2)

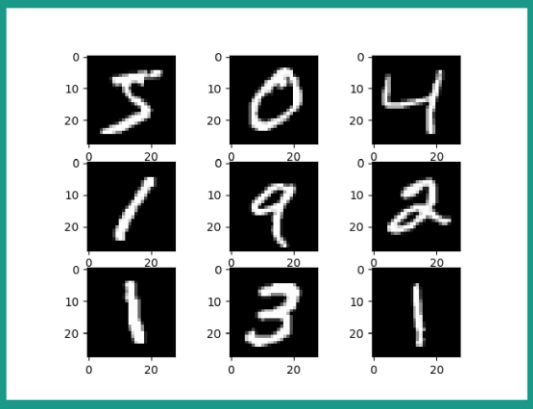


Model accuracy: 98.89%

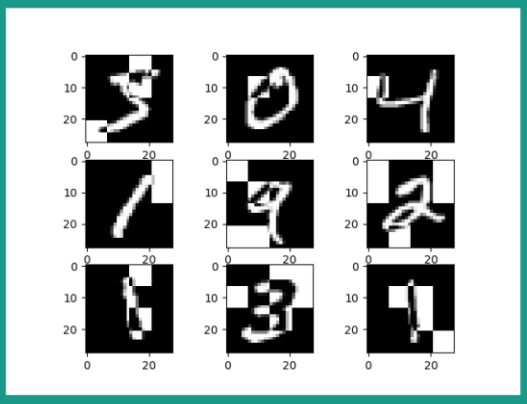


Model accuracy: 98.55%

Results (k=7, MNIST, Pixel Flipping)

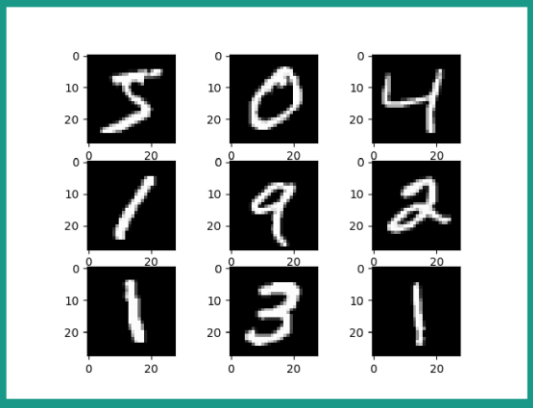


Model accuracy: 98.89%

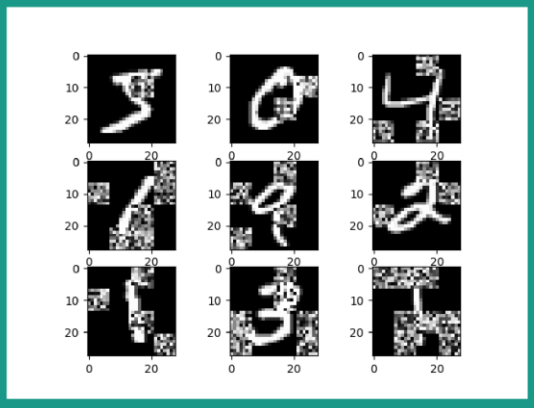


Model accuracy: 98.27%

Results (k=7, MNIST, Pixel randomization)

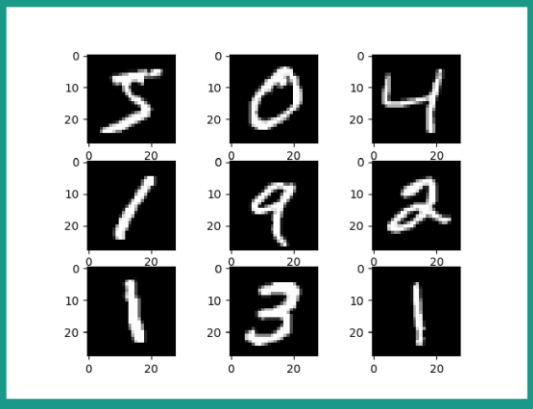


Model accuracy: 98.89%

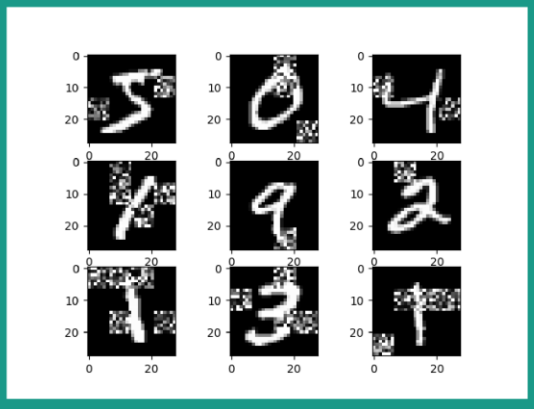


Model accuracy: 98.54%

Results (k=7, MNIST, Laplacian noise with epsilon 2)



Model accuracy: 98.89%



Model accuracy: 98.60%

Applying RR mechanisms to Images and studying its effect on downstream processes (continued)

Prajjwal
Pradeep
Abhin

Pixel Randomization Techniques Used

For a given block of K X K pixels, we first create a mask using one of the techniques below, then add the mask to the block followed by taking the absolute value and clipping to the range 0-255.

Pixel Flipping

Since pixel values range from 0-255, for a pixel value i, it's flipped pixel is the absolute value |i - 255|.

The mask is a matrix with all values as -255.

Pixel Randomization

Pixel values are randomized by sampling a random number between -255 and 255, adding it to the pixel, taking its absolute value and clipping it's value to 0-255.

The pixel values are directly not replaced by a random value because we are following the concept of application of a mask

Laplacian Noise Addition

A random sample from a laplacian distribution with location 0 and scale 255/epsilon is used to create the mask.

In addition to those :

- **Iterate the KxK in an overlapping manner by moving 1 pixel further after every iteration**
- **Adding Gaussian blur to the K x K block instead of proceeding without any perturbation when heads appear during a coin toss.**

The code
(perturbation)

```
def perturb(self, block) -> Any:
        """
        Perturbs a block of K X K pixels
        """
        mask = None
        if self.mode == "flip":
            mask = -255

        elif self.mode == "random":
            mask = np.random.randint(-255, 255, size=(self.K,
self.K))

        elif self.mode == "dp":
            with open('config.json') as config_file:
                configs = json.load(config_file)
            epsilon = configs["epsilon_dp_flip"]
            mask = np.random.laplace(loc=0, scale=255/epsilon,
size=(self.K, self.K))

        perturbation = block + mask
        perturbation = np.abs(perturbation)
        perturbation[perturbation > 255] = 255
        perturbation[perturbation < 0] = 0
        return perturbation
```

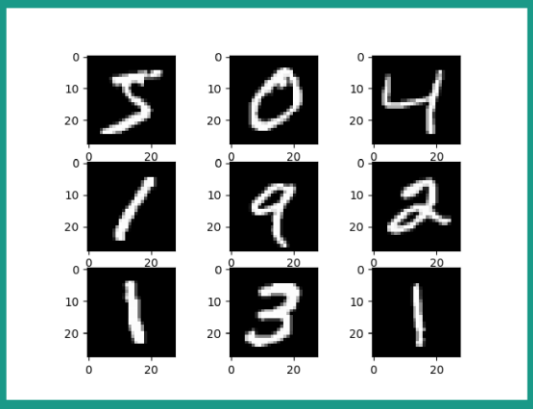
The code:
(iterating with overlap)

```
widthItr = (image.shape[1] )
heightItr = (image.shape[0] )

for i in range(widthItr-self.K):
    for j in range(heightItr-self.K):
        # TODO: Take care of edge cases
        block = image[j:j+self.K, i:i+self.K]
        do_i_perturb = np.random.rand()
        if do_i_perturb < configs["flip_prob1"]:
            do_i_really_perturb = np.random.rand()
            if do_i_really_perturb < configs["flip_prob2"]:
                image[j:j+self.K, i:i+self.K] = self.perturb(block)
            else:

                pass
        else:

            pass
  return image
```

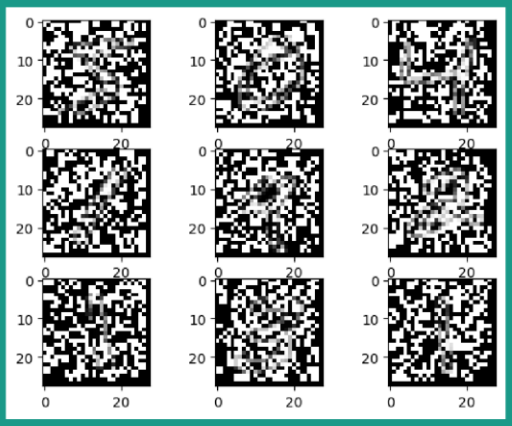
The code :
(Gaussian blur addition without overlap)

```
    # Convert the image array to float32 data type
image = image.astype(np.float32)


widthItr = (image.shape[1] // self.K)
heightItr = (image.shape[0] // self.K)

for i in range(widthItr):
    for j in range(heightItr):
        # TODO: Take care of edge cases
        block = image[j*self.K:(j+1)*self.K, i*self.K:(i+1)*self.K]
        do_i_perturb = np.random.rand()
        if do_i_perturb < configs["flip_prob1"]:
            do_i_really_perturb = np.random.rand()
            if do_i_really_perturb < configs["flip_prob2"]:
                image[j*self.K:(j+1)*self.K, i*self.K:(i+1)*self.K] = self.perturb(block)
            else:
                image[j*self.K:(j+1)*self.K, i*self.K:(i+1)*self.K]=gaussian_filter(block, sigma=1)



        else:
            image[j*self.K:(j+1)*self.K, i*self.K:(i+1)*self.K]=gaussian_filter(block, sigma=1)

return image
```

Results (k=2, MNIST, Pixel Flipping,overlap)

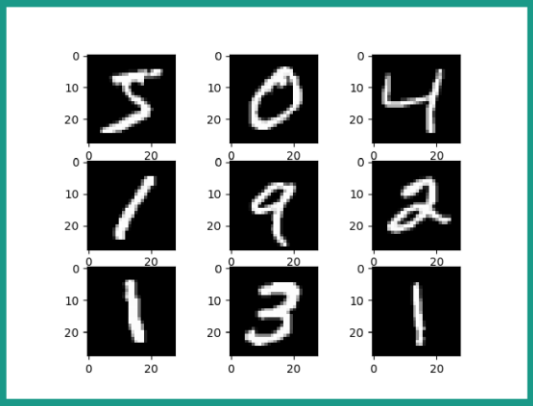


Model accuracy: 98.85%

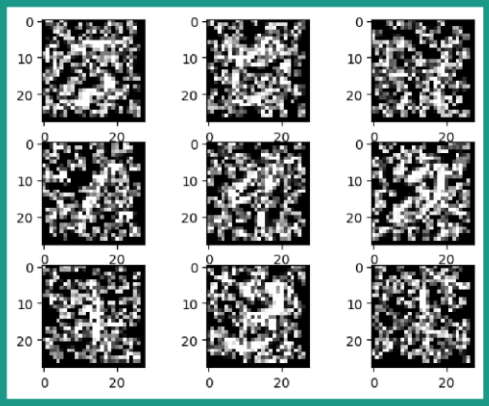


Model accuracy: 9.80%

Results (k=2, MNIST, Pixel Randomization,overlap)

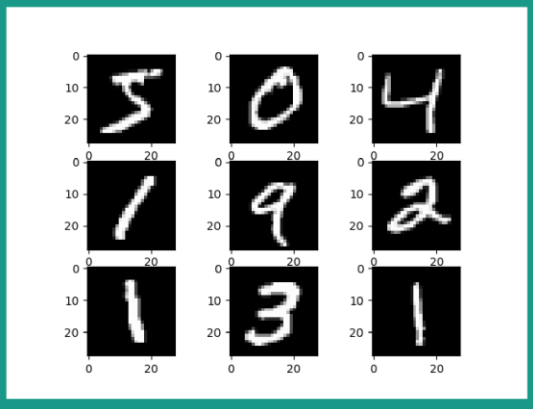


Model accuracy: 98.85%

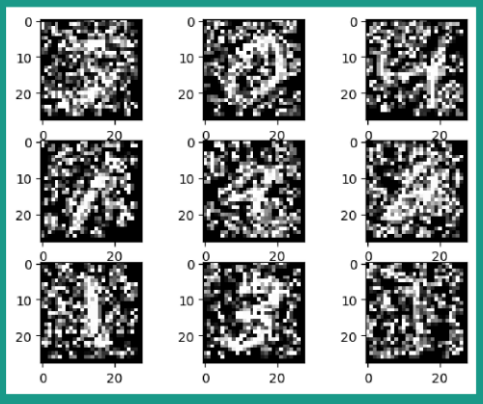


Model accuracy: 10.480%

Results (k=2, MNIST, Laplacian noise with epsilon 2, overlap)

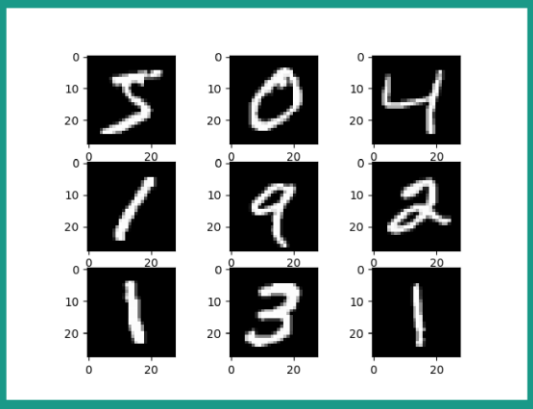


Model accuracy: 98.85%

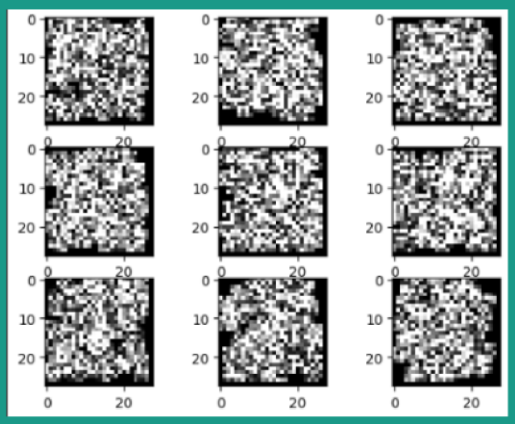


Model accuracy: 95.720%

Results (k=4, MNIST, Pixel Randomization,overlap)

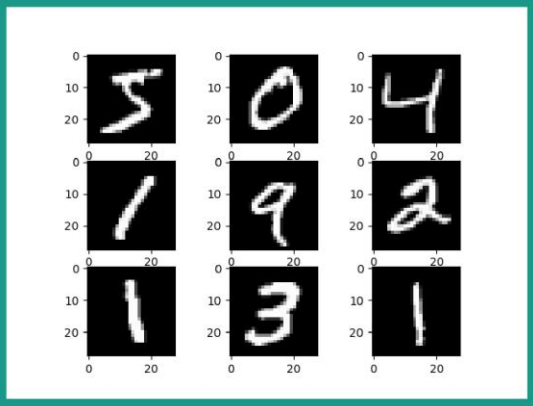


Model accuracy: 98.85%

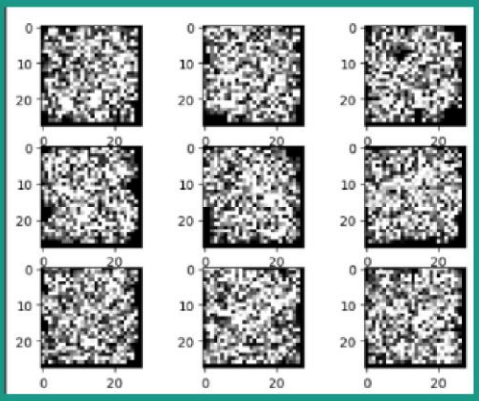


Model accuracy: 10.800%

Results (k=4, MNIST, Laplacian noise with epsilon 2, overlap)

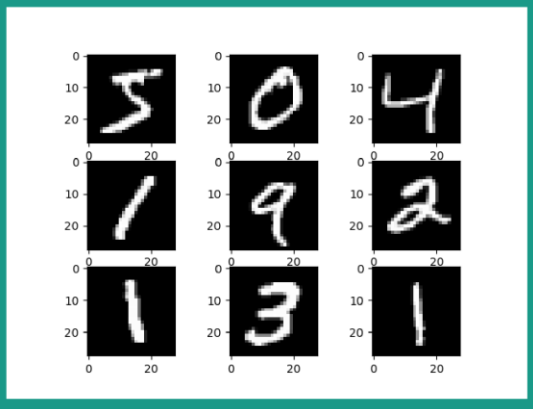


Model accuracy: 98.85%

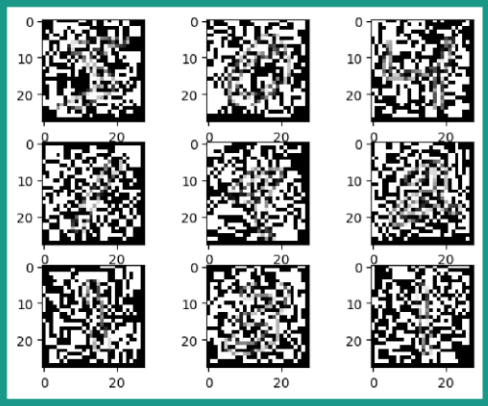


Model accuracy: 9.140%

Results (k=7, MNIST, Pixel Flipping,overlap)

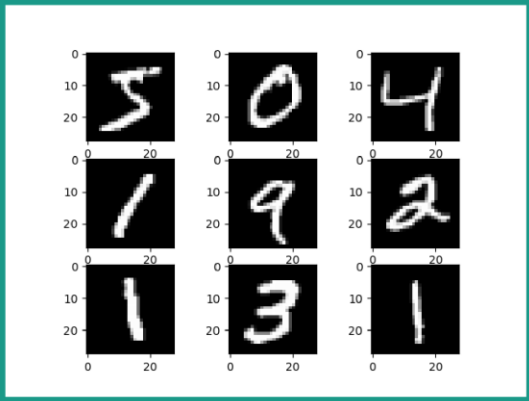


Model accuracy: 98.85%

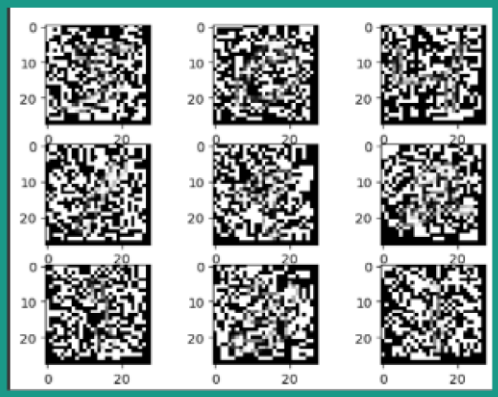


Model accuracy: 8.980%

Results (k=4, MNIST, Pixel Flipping,overlap)

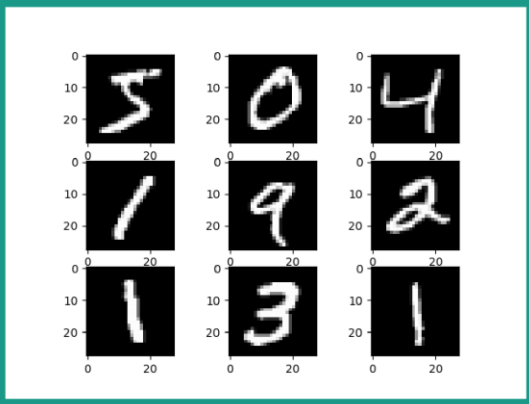


Model accuracy: 98.85%

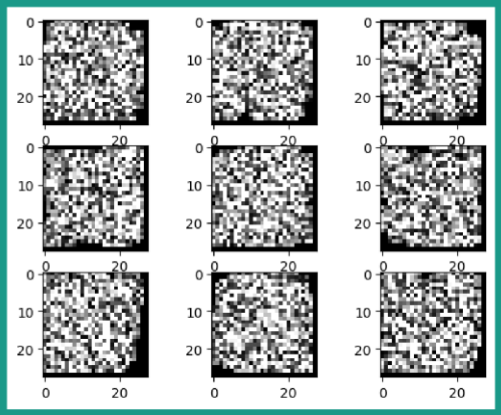


Model accuracy: 11.020%

Results (k=7, MNIST, Pixel Randomization,overlap)

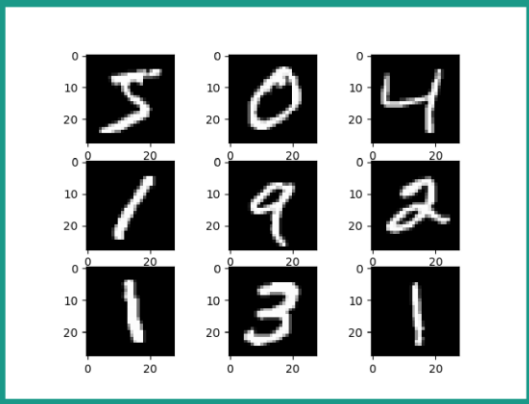


Model accuracy: 98.85%

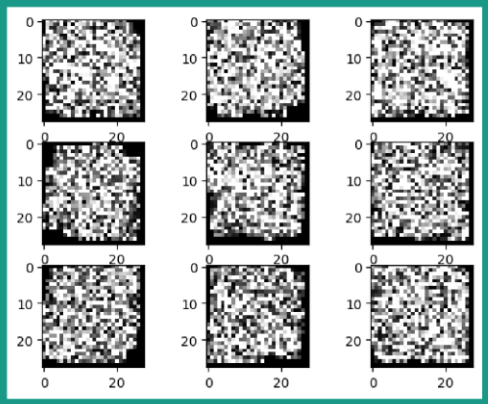


Model accuracy: 13.60%

Results (k=7, MNIST, Laplacian noise with epsilon 2, overlap)

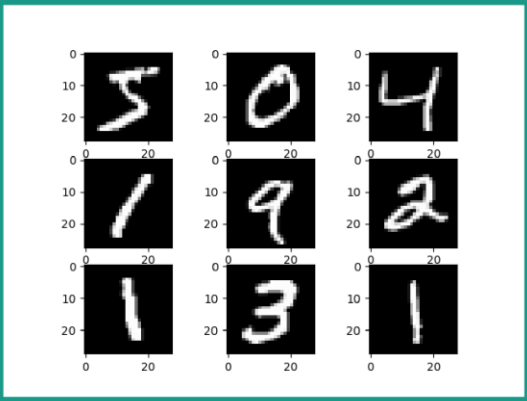


Model accuracy: 98.85%

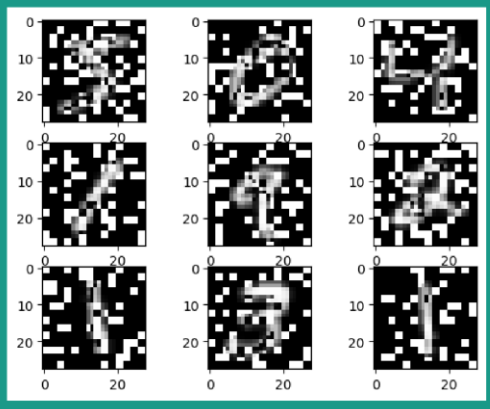


Model accuracy: 6.160%

Results (k=2, MNIST, Pixel Flipping, no overlap Gaussian Blur sigma=1)

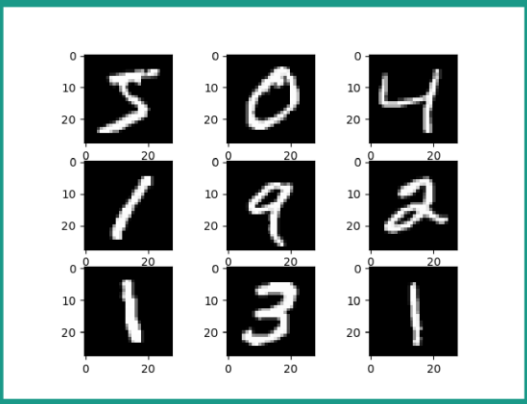


Model accuracy: 98.85%

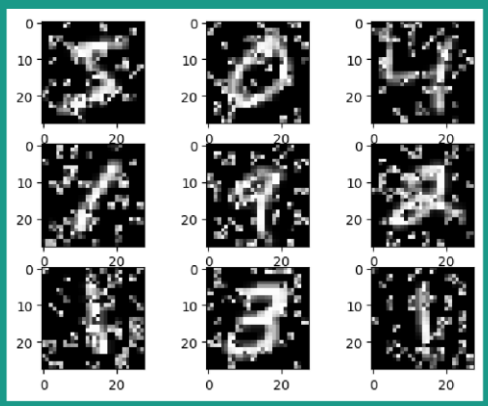


Model accuracy: 83.240%

Results (k=2, MNIST, Pixel Randomization, no overlap Gaussian Blur sigma=1)

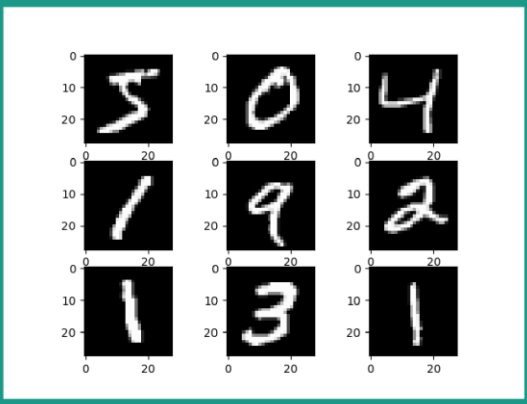


Model accuracy: 98.85%

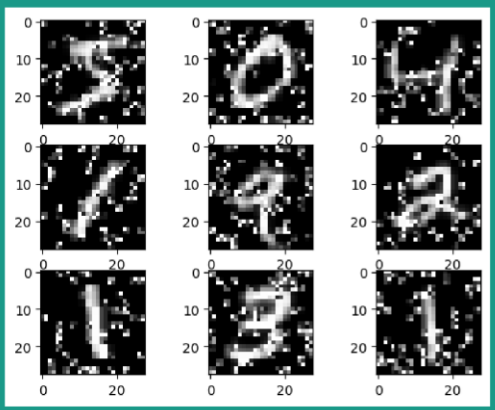


Model accuracy: 97.430%

Results (k=2, MNIST, Laplacian noise with epsilon 2, no overlap,Gaussian Blur sigma=1)

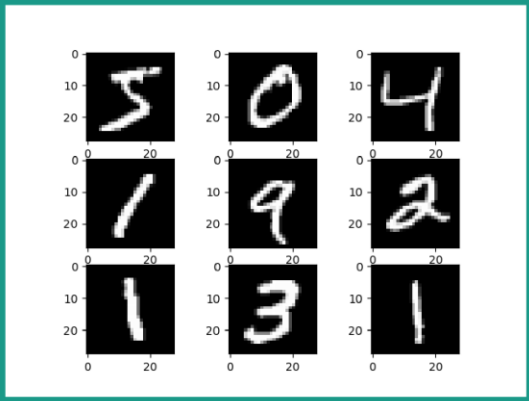


Model accuracy: 98.85%

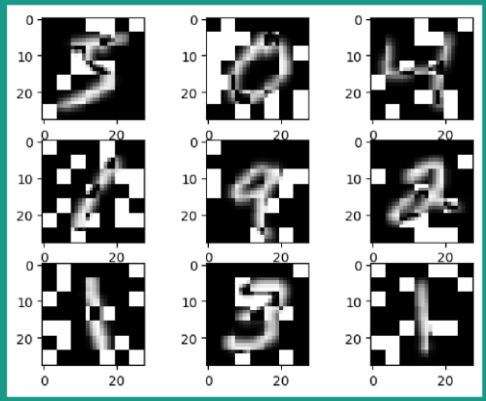


Model accuracy: 97.820%

Results (k=4, MNIST, Pixel Flipping, no overlap Gaussian Blur sigma=1)

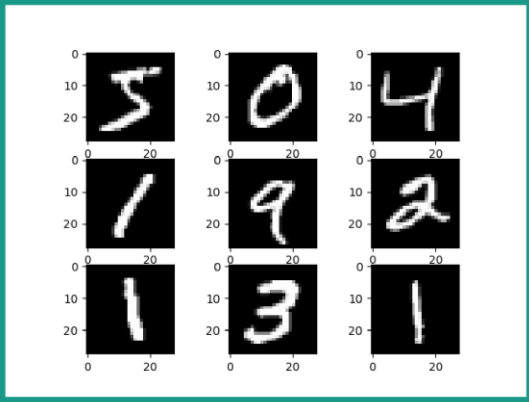


Model accuracy: 98.85%

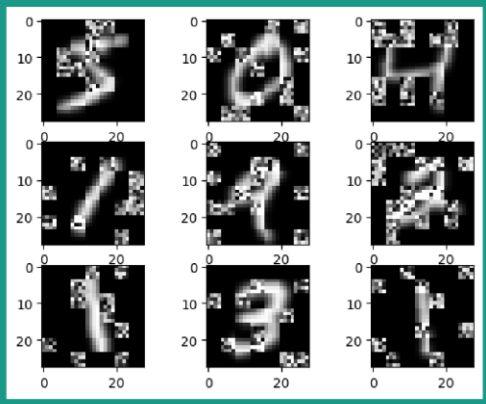


Model accuracy: 97.160%

Results (k=4, MNIST, Pixel Randomization, no overlap Gaussian Blur sigma=1)

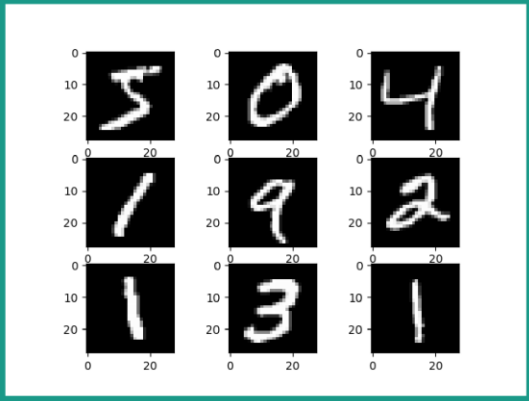


Model accuracy: 98.85%

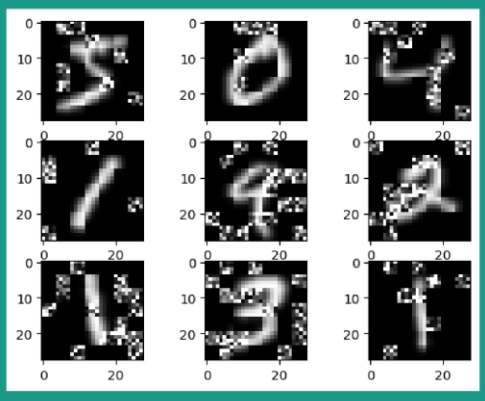


Model accuracy: 97.340%

Results (k=4, MNIST, Laplacian noise with epsilon 2, no overlap,Gaussian Blur sigma=1)

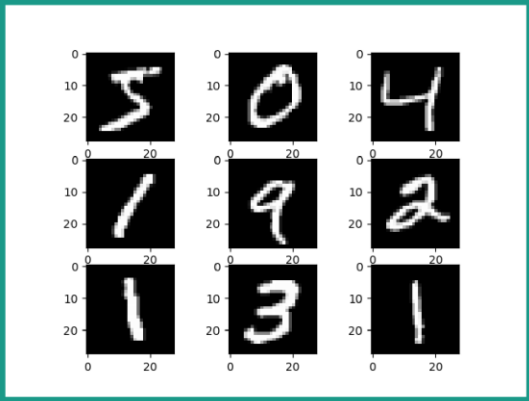


Model accuracy: 98.85%

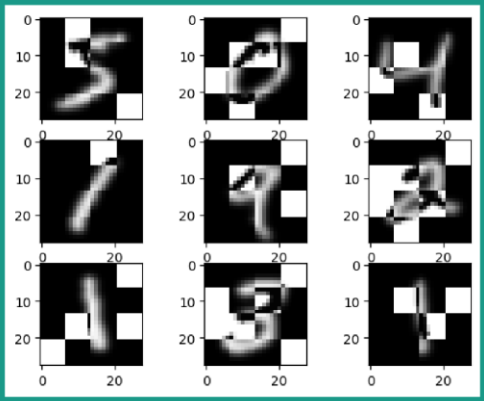


Model accuracy: 97.500.%

Results (k=7, MNIST, Pixel Flipping, no overlap Gaussian Blur sigma=1)

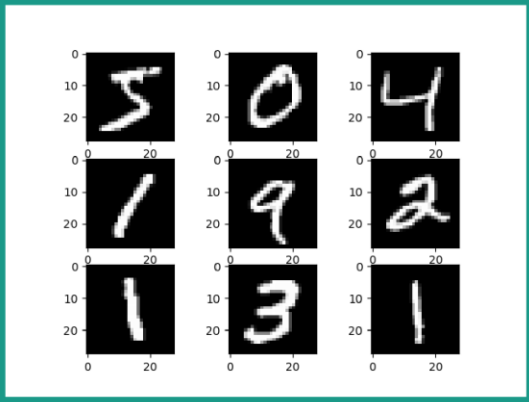


Model accuracy: 98.85%

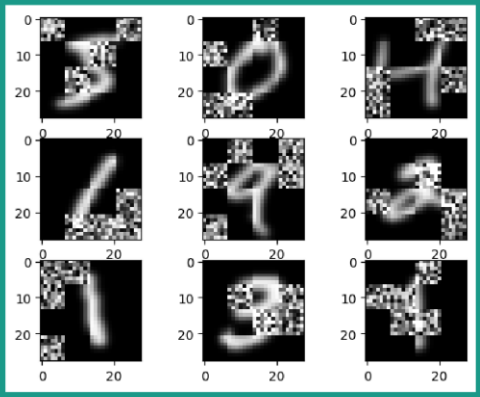


Model accuracy: 96.050%

Results (k=7, MNIST, Pixel Randomization, no overlap Gaussian Blur sigma=1)

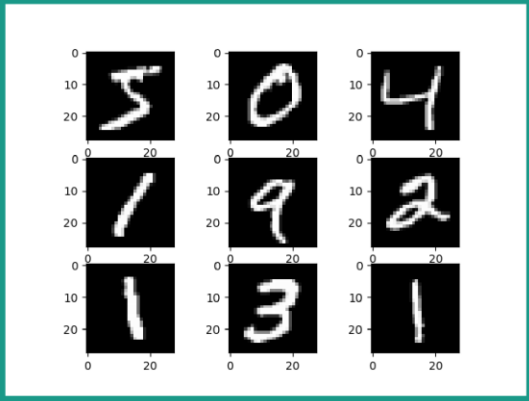


Model accuracy: 98.85%

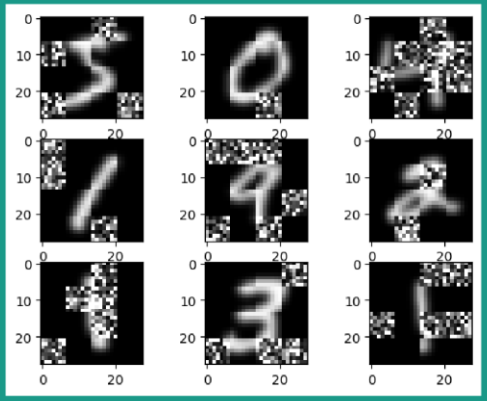


Model accuracy: 97.150%

Results (k=7, MNIST, Laplacian noise with epsilon 2, no overlap,Gaussian Blur sigma=1)

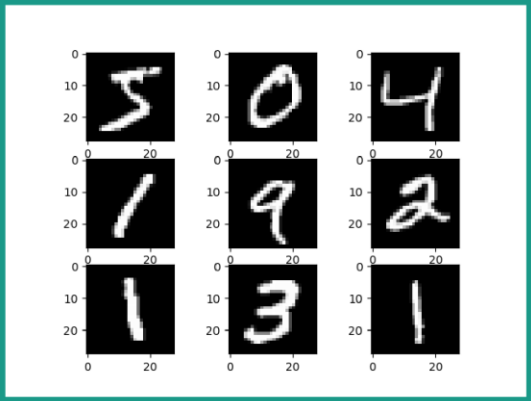


Model accuracy: 98.85%

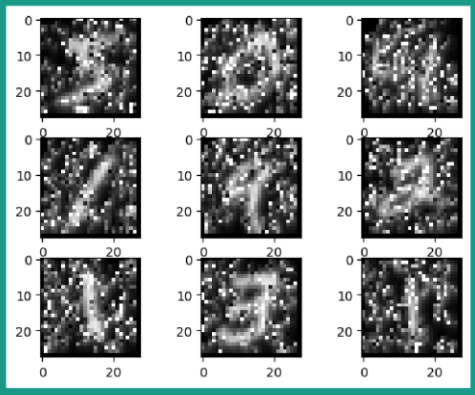


Model accuracy: 97.170%

Results (k=2, MNIST, Laplacian noise with epsilon 2, with overlap,Gaussian Blur sigma=1, flip prob=0.5)

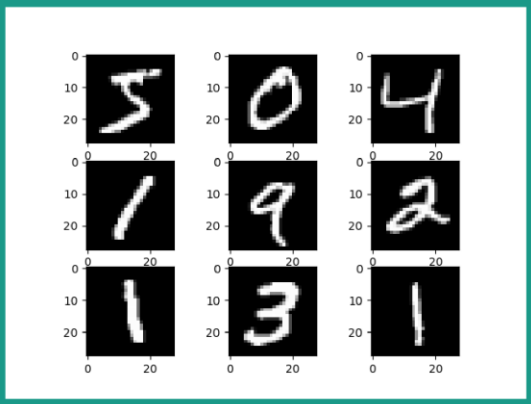


Model accuracy: 98.85%

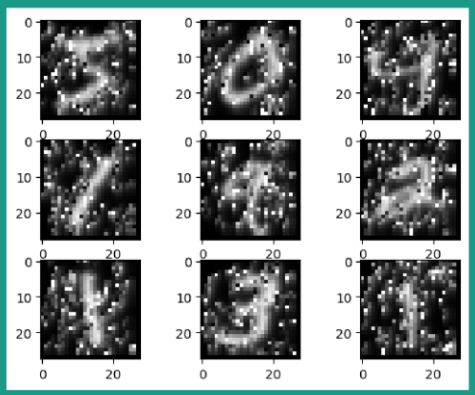


Model accuracy: 92.140%

Results (k=2, MNIST, Laplacian noise with epsilon 2, with overlap,Gaussian Blur sigma=1, flip prob=0.4)



Model accuracy: 98.85%



Model accuracy: 94.450%

Using Variational Autoencoders as PETs

Working of VAEs

- An autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process.
- Captures the latent space in form of two vectors representing mean and variance
- Used for generative purposes

What is to be passed downstream?

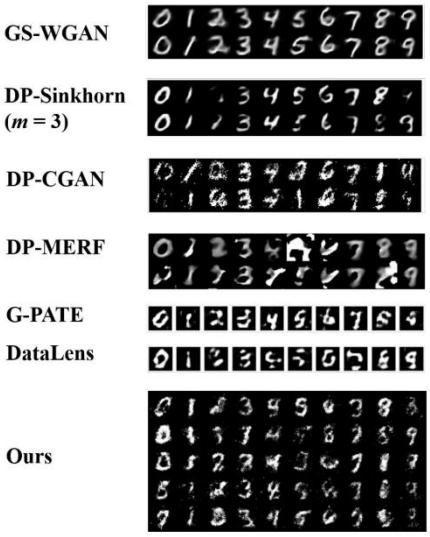
Encoded Samples	Generated Samples
- Autoencoders are enough - Changing the nature of data: In deployment also encoders will be required - Can privacy be violated by releasing encoders? - How do we handle object detection tasks?	- Need VAEs, GANs or Transformers - No need to pass down any models - How do we account for privacy of generated samples? - What if any actual images are generated with extremely high similarity - possible but less likely - How do we create annotations for generated image for object detection tasks?

Modifications for VAEs

Paper: **DP2-VAE: Differentially Private Pre-trained Variational Autoencoders**

Use modifications and pre training approaches for VAEs

Method	ϵ	MNIST			
		FID ↓	LR Acc ↑	MLP Acc ↑	CNN Acc ↑
Real data	∞	1.6	92.2	97.5	99.3
DP-CGAN	10	179.2	60	60	63
DP-MERF	10	121.4	79.1	81.1	82.0
G-PATE	10	150.6	N/A	N/A	80.9
DataLens	10	173.5	N/A	N/A	80.7
GS-WGAN	10	61.3	79	79	80
DP-Sinkhorn ($m = 1$)	10	61.2	79.5	80.2	83.2
DP-Sinkhorn ($m = 3$)	10	55.6	79.1	79.2	79.1
Ours	10	134.3	78.4	77.8	81.2



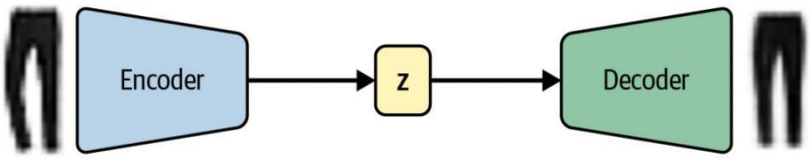
Can we do better?

- Can apply training algorithm agnostic approaches
- Can we estimate the maximum effect of a particular sample on the mean and variance vectors? : If yes, can we use it to define sensitivity for a DP based mechanism where a query is a generated sample? Will adding noise to the mean/variance vectors be a plausible privacy preserving mechanism?
- How do we put constraints on the maximum closeness of a generated vector in the latent space to an encoded vector from training space? If any constraints like this can be made -> No need of adding any noise?!

Privacy preserving image generation using Autoencoders with quantized latent space

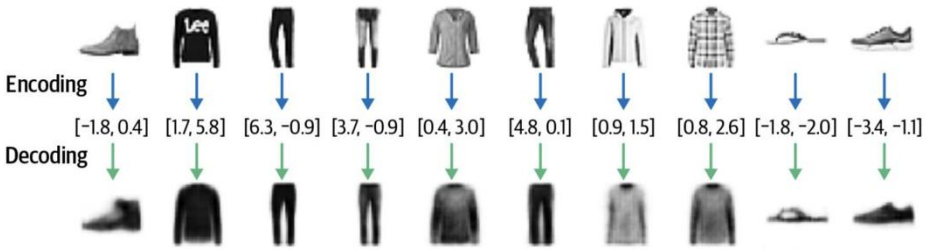
The Autoencoder

Simply encodes the input into one vector and attempts to reconstruct the input from the encoding.



Latent space for a simple autoencoder

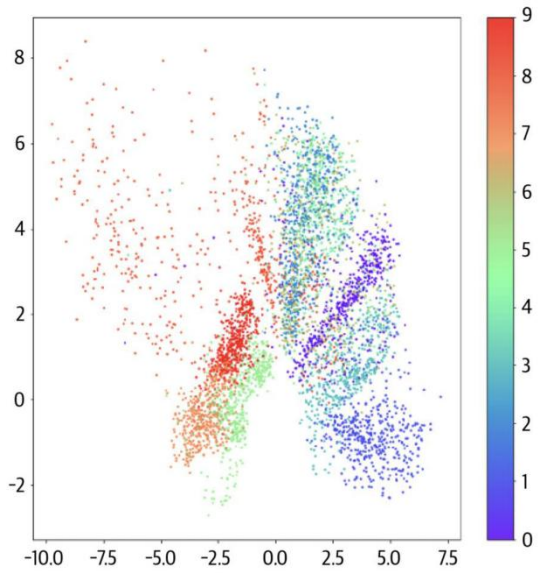
The latent space is spanned by a single dimension vector of length/size n (encoding dimension) where each component can take any value in R

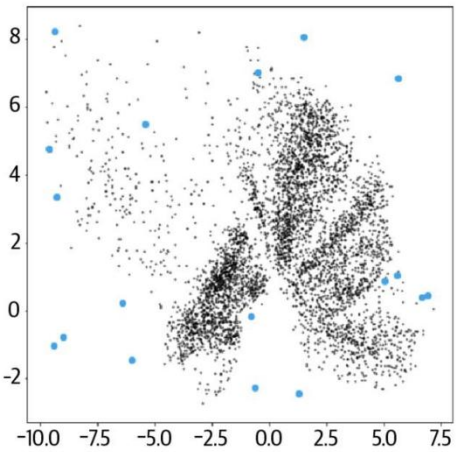


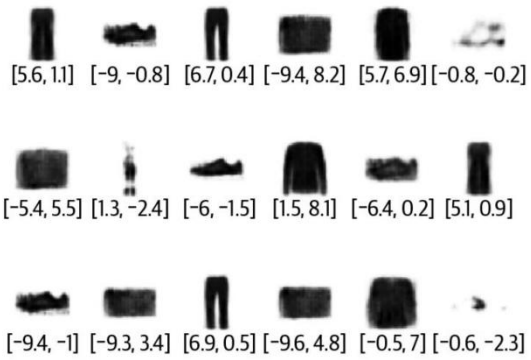
As a result, in the image generation step, the vector from which an image is to be generated can be a point in the latent space which is very far from training data.

This hurts our "quality" or "utility" of generative model.

Furthermore, the likelihood of generating a sample from the training set is same as generating any other sample in the latent space.





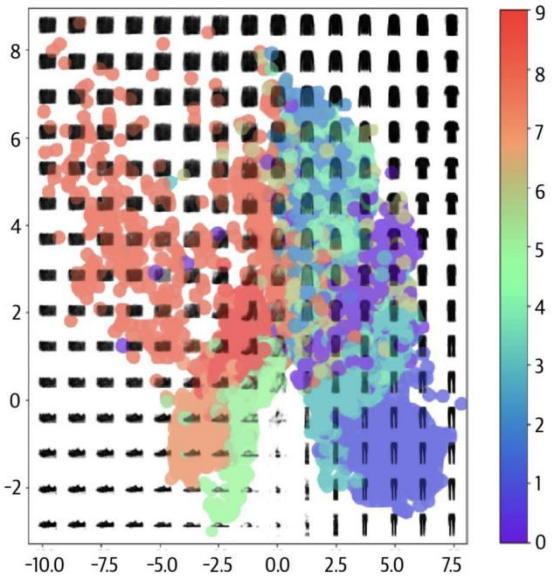


Privacy Model?

Ideally, we want to constrain the model such that the decoder does not generate any image which is very "close" or "similar" to any image in the training data.

How do we measure similarity between two images? (CV based techniques)

How do we constrain the model? -> Our novelty



Latent space quantization as a solution

Let's say we have some bounds (corners) to our latent space. If we are allowed to have "m" blocks in our space, we divide the entire bounded space into these m blocks.

After encoding any sample, we quantize the encoding to the nearest block in the quantized space.

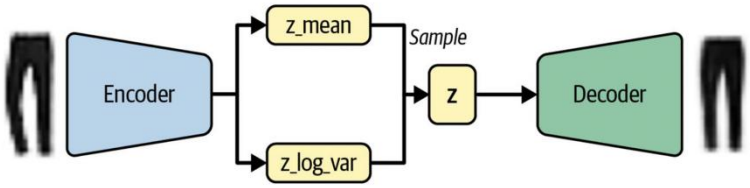
Result: Very similar images will have the same encoding!

Problem: Quantization of the latent space is a difficult problem!

Variational Autoencoder

Captures the latent space in two vectors: a mean vector, and a log variance vector.

Samples are generated from these two vectors



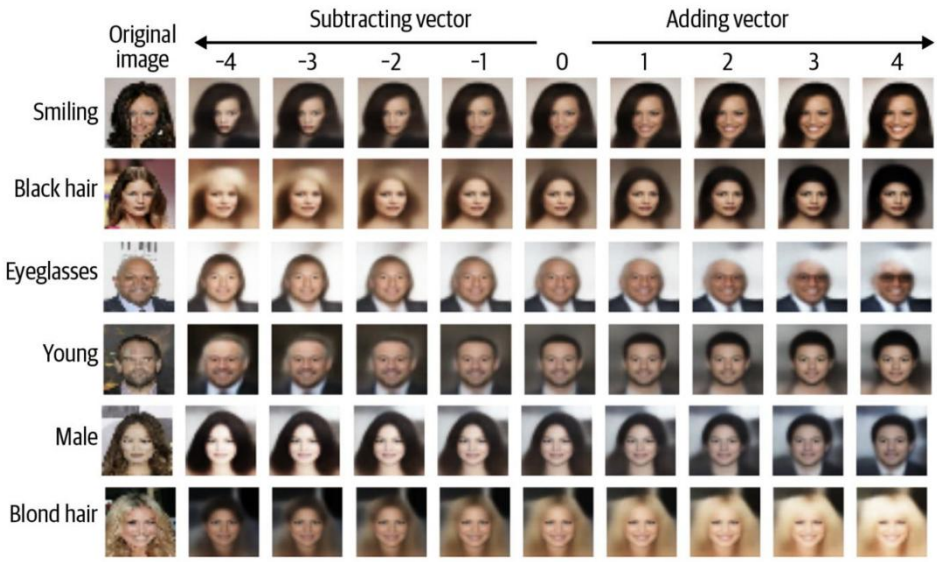
Latent space of VAE

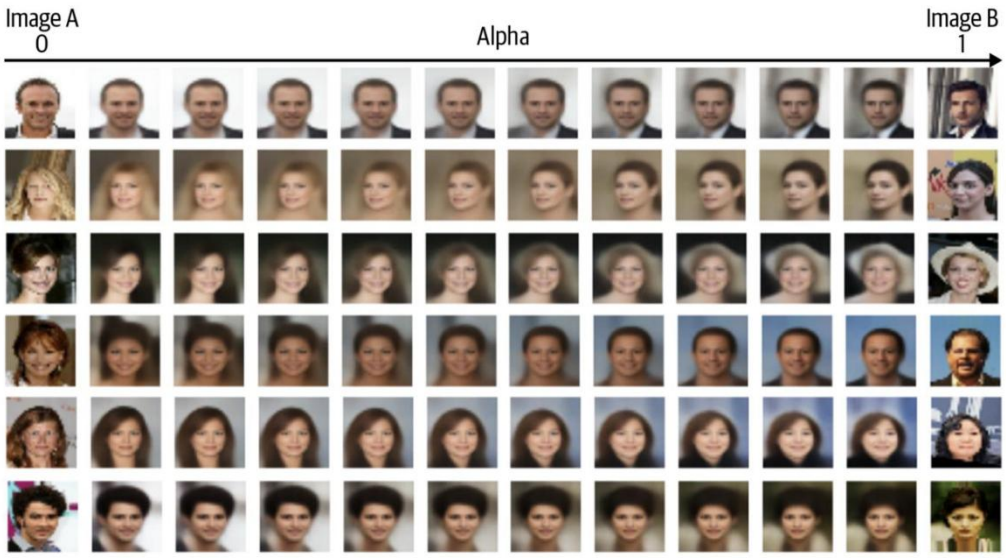
We sample a random number "epsilon" from a normal distribution with variance (sigma^2). The vector is generated by mean + (sigma^2 * epsilon).

With high probability (~99.7%), our vector will be mean +/- 3 sigma.

This encoded vector is passed to the decoder for reconstruction.

Result: "Smoother" latent space, allows interpolation between two classes/images.





Quantization of latent space for VAE

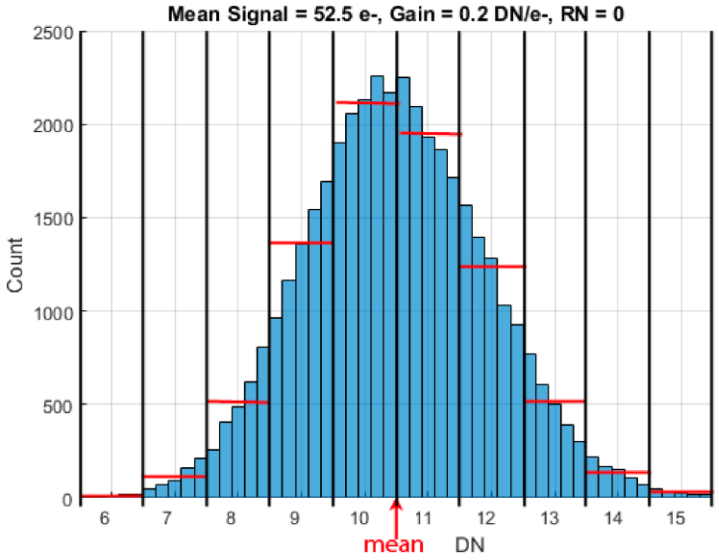
Possible solution:

Use a quantized gaussian/normal distribution to sample “epsilon”

-> Neighbouring vectors will have same probability of being sampled.

Problem: Will have to work on techniques to incorporate this in the training procedure. Original VAE used KL-Divergence loss for learning the mean and log variance vectors. But we cannot directly use KL-Divergence.

Novelty: New modified KL-Divergence loss which results in a VAE having privacy preserving properties



Personal Identifiable Information (PII):

- PII stands for **Personally Identifiable Information**. It refers to any information that can be used to identify or distinguish an individual.
  - that directly identifies an individual(e.g., name, address, social security number or other identifying number or code, telephone number, email address, etc.)
  - Or when combined with other data, can be used by an organization to identify specific individuals. In simpler terms, data that, when connected with other information, helps an agency figure out exactly who someone is. (These data elements may include a combination of gender, race, birth date, geographic indicator, and other descriptors).
- PII is sensitive information that, if exposed or mishandled, can pose risks to an individual's privacy, security, and potentially lead to identity theft, fraud, or unauthorized access to personal accounts or resources.
- The loss of PII can result in substantial harm to individuals, including identity theft or other fraudulent use of the information.

PII in various domains

- **Healthcare:** PII in the healthcare domain could include a patient's name, address, date of birth, medical record number, health insurance information, and any other personal details collected during the course of healthcare services.
- **Financial:** PII in the financial domain could include a customer's name, address, social security number, bank account number, credit card information, and other financial details collected by banks, credit card companies, or financial institutions.
- **Education:** PII in the education domain could include a student's name, address, date of birth, student identification number, academic records, and any other personal information collected by educational institutions.

- **Employment:** PII in the employment domain could include an employee's name, address, social security number, date of birth, employment history, and any other personal details collected by employers for employment-related purposes.
- **E-commerce:** PII in e-commerce could include a customer's name, address, phone number, email address, payment card information, purchase history, and any other personal details collected during online transactions.
- **Social media:** PII in social media platforms could include a user's name, email address, phone number, date of birth, and any other personal information provided by the user during the account creation process or while using the platform.
- **It is important to note that these are just a few examples, and PII can vary across different industries and domains.**

What was observed during the knowledge distillation process, specifically in terms of privacy leakage from the teacher model to the student model? Or does the student model have the ability to access sensitive data from the teacher model?"

Swing Distillation: A Privacy-Preserving Knowledge Distillation Framework: As per the paper says→

They have conducted distillation experiments on the AESLC dataset, which is a dataset for email subject line generation. They divide the dataset into two parts, one for the teacher model and the other for the student model. There is no overlap on the names of email senders in these two parts. After conducting knowledge distillation , they find that even if the names of email senders in the teacher dataset are not present in the student dataset at all, the student model after distillation still has a high probability to output these names. This clearly suggests that KD suffers from privacy leakage from the teacher model to the student model

Knowledge distillation code

https://colab.research.google.com/github/keras-team/keras-io/blob/master/examples/vision/ipynb/knowledge_distillation.ipynb

Knowledge Distillation

Abhin B

What is Knowledge Distillation?

Knowledge distillation refers to the idea of model compression by teaching a smaller network, step by step, exactly what to do using a bigger already trained network. As smaller models are less expensive to evaluate, they can be deployed on less powerful hardware.

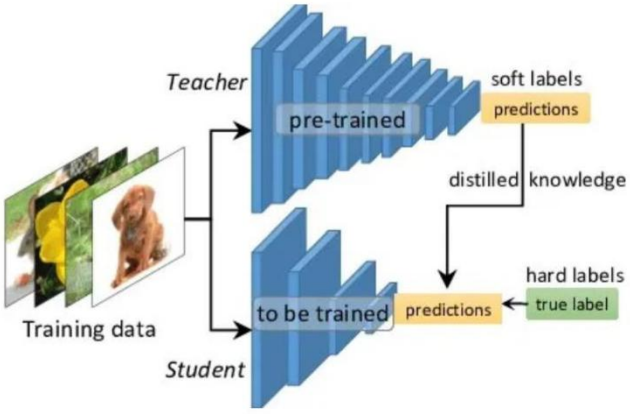
Why do we need this?

Deep Learning has achieved incredible performances in numerous fields including Computer Vision, Speech Recognition, Natural Language Processing etc.

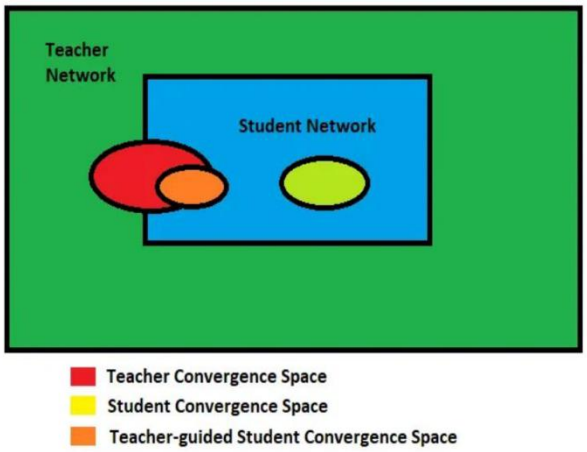
However, most of these models are too expensive computationally to run on devices like mobile phones or embedded devices.

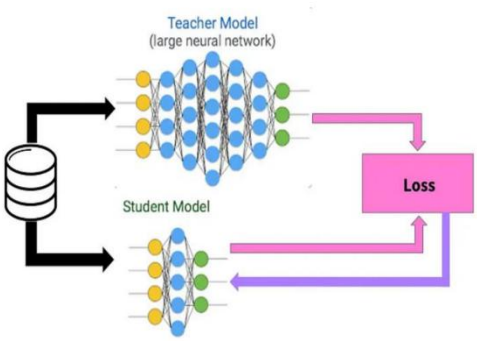
Knowledge distillation has been successfully used in several applications of machine learning such as object detection, NLP etc

Diagram



- The 'soft labels' refer to the output feature maps by the bigger network after every convolution layer.
- The smaller network is then trained to learn the exact behavior of the bigger network by trying to replicate outputs at every level





The student network can have a convergence which might be hugely different from that of the teacher network. However, if the student network is guided to replicate the behavior of the teacher network (which has already searched through a bigger solution space), it is expected to have its convergence space overlapping with the original Teacher Network convergence space.

Researching on **Privacy-Preserving Knowledge Distillation is a good approach to enhance knowledge transfer**

We know that Knowledge distillation is an elegant mechanism to train a smaller, faster, and cheaper student model that is derived from a large, complex teacher model.

There has been a massive increase in the adoption of knowledge distillation schemes for obtaining efficient and lightweight models for production use cases.

However, despite the success of KD, little effort has been made to study whether KD leaks the training data of the teacher model. KD actually suffers from the risk of privacy leakage

To address privacy concerns in knowledge distillation, it is crucial to acknowledge the potential risk of privacy leakage in the transfer of information from the teacher to the student model

Furthermore, privacy techniques like DP can play a significant role in enhancing privacy during the knowledge distillation process.

By incorporating these techniques, such as injecting noise into soft targets, perturbing the training objective, or employing privacy-preserving aggregation methods, formal privacy guarantees can be established. These measures effectively preserve privacy while transferring knowledge from the teacher to the student model.

Examples where privacy concerns are motivates in KD

Medical Diagnosis: A teacher model trained on medical records containing sensitive patient information, including diagnoses, treatments, and potentially personally identifiable information (PII).
Privacy Concern: If the student model directly learns from the teacher model without proper privacy-preserving techniques, it may inadvertently expose patients medical history, conditions, or PII, violating their privacy rights.

Financial Fraud Detection: A teacher model trained on a financial dataset containing transaction details, including account numbers, transaction amounts, and timestamps.
Privacy Concern: If the student model is not trained with appropriate privacy safeguards, it could potentially reveal sensitive financial information, leading to privacy breaches and potential misuse of individuals financial data.

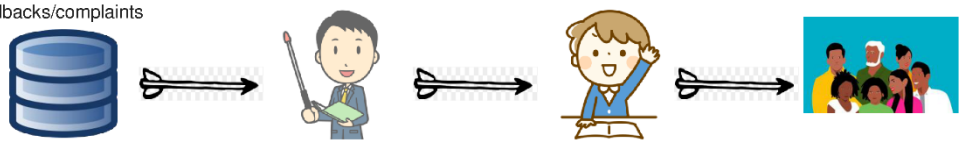
Social Media Analysis: A teacher model trained on a social media dataset containing users posts, comments, and engagement information.
Privacy Concern: If the student model learns from the teacher model without privacy protections, it could unintentionally expose users personal opinions, preferences, social connections, or private information shared on social media platforms.

Online Behavior Tracking: A teacher model trained on a dataset tracking users online behavior, such as browsing history, search queries, and interactions with websites or advertisements.
Privacy Concern: If the student model is not trained with privacy-preserving techniques, it could potentially reveal users browsing habits, personal interests, or potentially sensitive information that was part of their online activities

Example: Biases in the Teacher

Example 1: A teacher model with a dataset of feedback for a specific product

The dataset contains a mix of feedbacks/complaints



During the knowledge distillation process, the teacher model transfers its knowledge to a student model. However, the teacher mode is biased. As a result, the student model also mimics this while learning. When the student model is deployed and interacts with customers, by observing the student model's behaviour and responses, an external observer or adversary might infer that the teacher model is biased.

Whose privacy are we preserving?

Let's look at some cases.

- Case 1: In the previous scenario, this inference has the potential to expose privacy concerns or vulnerabilities related to the teacher model and the training dataset.
- Case 2: In the situation where each customer is personally sending their feedback to the teacher, the objective is to safeguard the privacy of each individual.
- Case 3: When a contributor provides data to the teacher, the aim is to ensure the privacy protection of that contributor.

Example 2 : A teacher having a dataset of customer purchasing behaviors for a specific online store

- The dataset contains information about the products customers have purchased, their preferences, and their demographic attributes.
- However, due to biases in the training data or the modeling process, the teacher model inadvertently focuses more on customers from a specific demographic group, such as young adults.
- As a result, the student model also exhibits a bias towards the preferences and behaviors of young adults.
- When the student model interacts with customers, it tends to recommend products or make predictions based on the preferences of young adults, potentially neglecting the needs and interests of other demographic groups.
- From the recommendations or predictions made by the student model, an external observer or customer might infer that the teacher model had a bias towards young adults.
- This inference could reveal information about the training data used by the teacher model and potentially expose privacy concerns or limitations in the model's understanding of diverse customer demographics.

Research paper related to Knowledge distillation

- Swing Distillation: A Privacy-Preserving Knowledge Distillation Framework

  https://arxiv.org/pdf/2212.08349.pdf

Knowledge Distillation (cats_vs_dogs)

Data-set :

- The Cats vs. Dogs dataset is a standard CV dataset that involves classifying photos as either containing a dog or cat.
- The dataset consists of 25,000 images with equal numbers of labels for cats and dogs, with 20,000 images in the training set and 5,000 images in the test set
- The label for cat in the dataset is 0. The label for dog in the dataset is 1

Observations:

Epochs : 20

Teacher

- Accuracy: 91%
- Total test data: 5000
- Cats classified correctly: 2270
- Dogs classified correctly: 2730

Distilled student

- Accuracy: 89.4%
- Total test data: 5000
- Cats classified correctly: 2235
- Dogs classified correctly: 2765

Student trained from scratch

- Accuracy: 86.8%
- Total test data: 5000
- Cats classified correctly: 2170
- Dogs classified correctly: 2830

- The results still show that knowledge distillation can be an effective way to improve the accuracy of a smaller model trained on a limited dataset.
- The teacher model is able to achieve a high accuracy of 91%, because the teacher model is able to learn the underlying patterns in the data and generalize well to new images.
- The distilled student model is able to achieve an accuracy of 89.4%, which is slightly lower than the accuracy of the teacher model. However, the distilled student model is still able to outperform the student model trained from scratch, which achieves an accuracy of 86.8%.

Will the student model be able to memorize the data?

- The distilled student model correctly classified 2235 cats and 2765 dogs, while the teacher model correctly classified 2270 cats and 2730 dogs.

Is this enough to conclude that the model is memorising the data?

So from that we can only say that the student model may memorise the data.

- Another way to check if the student model is memorizing the data is to use a technique called adversarial training. With adversarial training, we create a adversarial example for each input image in the training set. An adversarial example is a slightly perturbed version of the input image that is still classified correctly by the student model.
- If the model performs poorly on adversarial test examples, model has a high possibility of memorizing data

Adversarial test

- Epochs:10
- Dataset : MNIST dataset

```
Accuracy: 94.00%
Adversarial accuracy: 77.00%
```