Does AI Remember?
Neural Networks and the Right to be Forgotten

Laura Graves , **Vineel Nagisetty** , **Vijay Ganesh**
University of Waterloo
{laura.graves, vineel.nagisetty, vganesh}@uwaterloo.ca

Abstract

The *Right to be Forgotten* is part of the recently enacted General Data Protection Regulation law that affects any data holder that has data on European Union residents. It gives EU residents the ability to request deletion of their data. This includes training records used to train any machine learning model that data holders might own. In particular, deep neural network models are vulnerable to model inversion attacks which extract class information from a trained model. If a malicious party can mount an attack and learn private information that was meant to be forgotten, then it implies that the model owner has not properly protected their user's rights and may not be compliant with the General Data Protection Regulation law. We present a general threat model to show that simply removing training data is insufficient to protect users. We further propose and evaluate three defense mechanisms (deemed *neuron removal, scattered unlearning,* and *class unlearning*) that could help model owners protect themselves against such attacks while being compliant with regulations. We show that these defense mechanisms enable deep neural networks to *forget* sensitive data from trained models while maintaining model efficacy. A copy of our code, which can be used to replicate our results, can be found at http://tiny.cc/forgetfulnet.

1 Introduction

In 2016 the European Union (EU) established the *General Data Protection Regulation* (GDPR) which is intended to allow individuals in EU nations control over their personal data. This includes regulations for businesses that handle personal data, requiring them to provide safeguards to protect data and use the highest-possible privacy settings by default [?]. In particular, article 17 of the GDPR gives individuals the Right to be Forgotten (RTBF) and states that ... "(businesses) have the obligation to erase personal data without undue delay" [?]. Individuals who invoke the Right to be Forgotten must have their personal data removed from the requested datasets.

Unfortunately, the existence of model inversion (MI) attacks [?] that extract class information from trained models poses a unique problem for model owners with respect to GDPR. If an individual's data has been used to train a model and that individual invokes the RTBF, simply deleting the training data is not sufficient to be compliant with GDPR. This is because deep learning algorithms learn properties (including private data) of the dataset they are trained on. Consequently, if a malicious party can mount an attack and learn private information that was meant to be forgotten, then it is possible to conclude that the model owner has not properly protected their user's privacy rights.

A naive attempt by a model owner at solving the above-described problem may be to discard their trained model, remove the individual's data and train a new model from scratch. Sadly, training machine learning models can be a very expensive and time-consuming process. Model owners are likely to be disinclined to bring down their model, while a new model is being trained which is compliant with the request from a customer and the GDPR law. It is in the best interest of model owners to find ways to forget data that requires less time and resources while ensuring the fidelity of their network, and compliance with laws and policies set forth by governmental agencies. Humerick [?] highlights the legal importance of ensuring data is forgotten when requested (thus remaining compliant with RTBF), as well as the economic harm that could be caused by requiring models to be retrained from scratch.

Problem Statement: Our work focuses on the following question: if individuals or groups invoke the right to be forgotten, how can a model owner ensure the data is forgotten while minimizing cost?

1.1 Contributions

We make the following contributions in this paper:

1. We explore the problem of how to make DNNs compliant with the Right to be Forgotten policy of the GDPR law while minimizing cost to model owners. In this context, we introduce three different defense mechanisms that cause a neural network to efficiently forget learned data thus protecting model owners, without harming model performance and at a relatively low cost. We further compare these methods against the naive method of

removing the data and retraining. The defense mechanisms we introduce are: neuron removal (directly removing the output layer neuron corresponding to the forgotten class, with optional retraining for a small number of iterations), scattered unlearning (randomly labeling data with the forgotten class label, before retraining for a moderate number of iterations) and class unlearning (random relabeling of the forgotten class examples, before retraining for a very small number of iterations). For a full explanation of these methods see section 4.

2. We provide detailed comparisons of the above-mentioned defense mechanisms, specifically through metrics such as number of retraining iterations (cost) and model accuracy. Further, we evaluate the efficacy of our defense mechanisms against model inversion attacks. The ideal defense mechanism is one which is not only effective, but also requires very few retraining iterations and does not diminish the accuracy of the retrained model on non-forgotten data. One of the key findings of our work is that class unlearning defense mechanism is the best one among the three methods we evaluated. (Section 5). Additionally, as part of our effort to formalize an appropriate threat model in this context, we found that some modifications to the model inversion attack [?] result in an attack that is effective on convolutional neural networks previously considered impervious to such attacks. (Section 3.2)

In this work we use the terms defense mechanism and forgetting method interchangeably. Further, a defense mechanism is defined as any method that can be applied to DNNs to cause them to become resistant to a certain class of attacks. The space we explore here is methods that cause DNNs to be resistant to leaking information about a class. We want to note that we considered the question of providing a mathematical certificate that guarantees compliance of a DNN with RTBF. However, at this point in time we find this question of mathematical guarantees in the setting of DNNs to be too difficult, and instead focus on pragmatic defense mechanisms that can be evaluated relative to each other based on appropriate metrics or observations based on whether a class of attacks is successful (such as the ones we suggest in this paper).

2 Prior Work

Machine learning models have been shown to leak (private) information about their training data [?; ?; ?; ?; ?]. Specifically, two main kinds of information leakage attacks have been studied: *membership inference attacks* that leak information about specific records[?] and *model inversion attacks (MI)* that leak information about the classes[?; ?].

Membership inference attacks determine whether a particular record was used to train a model. This work was first highlighted by Homer et al. in [?] and formalized by Dwork et al. in [?]. Other works include attacks and defense mechanisms against membership inferences [?; ?; ?; ?; ?]. Property inference attacks are a subset of membership inference attacks that determine a general property of the training data, such as the ratio between classes [?]. Model inversion attacks, introduced by Fredrikson et al. in [?] and expanded to vision tasks in [?], have been shown to recreate instances of records from an ML model. Given access to a white box classifier, examples of target classes can be recreated. In our paper, we contribute modified versions of these inversion attacks to determine if models leak class information.

There is an abundance of literature on differential privacy which provides an upper bound on the amount of privacy leakage from each individual data record [?; ?; ?; ?]. Cummings and Desai 2018., address the need for training machine learning models in a differentially private manner to comply with GDPR[?]. However, differential private algorithms only ensure privacy of the records and do not enable data to be forgotten once requested (such as by an individual invoking the RTBF). By contrast, our work aims to infer whether an adversary can recover knowledge of a particular class, once a model has been modified to forget that class.

Other recent research has touched on the issue of removing learned properties from machine learning models. Ginart et al. [?] devised a notion they term *removal efficiency* and give two algorithms for efficiently removing specific data points from k-means clustering models. Guo et al. [?] defined an approach they term *certified removal* that was evaluated for linear classification models. In this system, a model is trained on a dataset including sensitive data, and then the sensitive data is removed in some way from the model. A different model is trained on the same dataset without the sensitive data, and removal algorithms are evaluated based on the difference between the models. However, these approaches are unlikely to work in the case of DNNs, which are more opaque and difficult to analyze.

Very recently, Bourtoule et al. [?] introduced a method for dealing with individual data removal requests. They proposed SISA training, a method consisting of an aggregate model made of multiple models trained on disjoint partitions of the data. Because of this segmentation, when requests for removal are made the model owner can retrain only the effected sub-models instead of having to retrain everything. By contrast, our method focuses on deep learning models that have already been trained. Further, we have no requirement to have the model be an aggregate of weak learners. We use MI attacks as a way of evaluating what information such models retain. Finally, we propose several defense mechanisms that model owners can use to forget learned data, and we provide detailed empirical evaluation of these mechanisms on cost and model accuracy. Unfortunately, the work presented in Ginart et al. [?] and Guo et al. [?] are both focused on such different methods that it is impossible to make a direct comparison with our work. We hope in the future more research is done on comparable models, because we believe that this area of research is still in its infancy.

3 Model Inversion Attacks and Threat Model

An MI attack is best explained with a motivating example. Consider the following scenario: an airport security chief has trained a facial recognition system to recognize individuals on their do-not-fly list. Each class represents an individual who has been identified as a potential security threat. An insider leaks a copy of the facial recognition system model to an attacker, but no other data. A motivated attacker who wishes to find out who's on the do-not-fly list can perform an MI attack on the leaked model to extract images of the individuals, compromising airport security. In this paper, we focus on similar situations where a model classifies individuals, with each class marking one unique individual. MI attacks were chosen because they can be used to extract class information from a trained model with only white box access to that model. We introduce several defense mechanisms that may be used by the model owner to forget a class. To test the effectiveness of these forgetting methods, we use Frederikson et al's MI algorithm [?] with some alterations that make it more effective against convolutional neural networks (CNNs) (see section 3.2).

3.1 Threat Model

Notation: In this work we use D to refer to the entire dataset a model M is trained on. c_i represents a class, while $\{c_i\}$ denotes the set of all examples belonging to that class.

In our threat model, a DNN M is trained on a dataset consisting of n classes $D = \{c_1, ..., c_n\}$. Each class $c_i \in D$ represents an individual. After training, the individual whose data belongs to class c_t invokes the right to be forgotten. The model owner takes action to make M forget c_t and we denote this new model as M'. An adversary gets white box access to M' and, using only the label information of the target, tries to extract information about c_t. We consider an attack successful if the attacker can generate an example that is "identifiable" as belonging to c_t. We assume the attacker has computational resources that are polynomial in the size of the DNN, which is sufficient to mount an MI attack.

3.2 Model Inversion Algorithm

The typical MI attack is a deterministic attack on a machine learning model to which the attacker has white box access. The algorithm starts with an example with all features at 0 (or a suitable starting point for the domain) and labels this blank example with the target class label c_t. The algorithm then iteratively processes the example through the model (in the forward pass) and calculates the gradient of the loss with regard to the example itself. The example is updated each iteration, altering it in a way such that it is closer to what the model considers to be that class. For a detailed explanation of the algorithm see [?].

The first alteration we make to the MI algorithm is to the *PROCESS* function that is performed each iteration after the gradient descent step. In the original algorithm this function helps make the inverted image recognizable by optionally performing some image processing steps. In their examples, PROCESS either does nothing or it applies a de-noising filter followed by a sharpening filter to the example. We implement a variant of this approach into our defense mechanism using function *refine()*. Instead of being applied every iteration, we apply this image refinement periodically during the inversion process. We found that this approach has the benefits of image refinement without washing out important features, unlike the case where the step is performed every iteration.

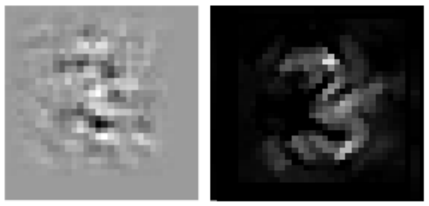
Figure 1: Left: MI attack from Fredrikson et al. Right: our modified MI attack

We also note that because the MI attack is a deterministic process, beginning each inversion from a static "blank" example results in the same inverted example each time. Instead, we start with a small amount of noise applied to each feature. This results in a variety of inverted images, with a slightly different result each time the inversion is performed. We found that larger amounts of noise resulted in noisier inverted images but were sometimes necessary to generate inversions from models with very small gradients.

Finally, the Frederikson et al. attack continues the attack process until loss starts increasing. We found that a significant amount of change happens even after the loss starts increasing, and we instead run the inversion attack for a set number of iterations regardless of change in loss.

An interesting observation we made is that these modifications result in an MI attack that is very successful even when applied to CNNs, a class of neural networks that are notoriously difficult for MI attacks. Hitaj et al. [?] explored this question and concluded "MI works well for MLP networks but clearly fails with CNNs". Not only were we able to perform MI attacks using our algorithm, but we did so on the architecture that was used by Hitaj et al. using the digit "3" as a target. The differences can be seen in figure 1.

4 Defense Mechanisms for DNNs

In this section, we discuss defense mechanisms and their properties. Recall that a defense mechanism is defined as any computational method that can be applied to DNNs to cause them to become resistant to a certain class of attacks. In this section we explore methods that cause DNNs to be resistant to leaking information about a class. To this end, we define a set of desirable properties that we believe defense mechanisms must possess:

- A defense mechanism should maintain the *quality* of the original model, not harming it's efficacy on any of the non-forgotten data. One way to measure the quality is by the model's test accuracy on the non-forgotten classes.
- A defense mechanism should also be *efficient* in terms of model training time or algorithm runtime.
- A defense mechanism should be *effective* at making models robust to inversion attacks on the forgotten class. We acknowledge that using metrics to evaluate efficacy of a defense mechanism is a difficult task and we are currently evaluating them based on observing inversion attacks.

We define four defense mechanisms that we evaluate based on the above-mentioned properties:

4.1 Defense Mechanisms

1. **Retraining:** we remove all data belonging to c_t from the dataset D to produce $D' = D \setminus \{c_t\}$ and continue training the model for some iterations on D'.
2. **Neuron removal:** we remove the output neuron corresponding to c_t. We then optionally remove all data belonging to c_t from the dataset D to produce $D' = D \setminus \{c_t\}$ and continue training the model for some iterations on D'.
3. **Scattered unlearning:** we remove all data belonging to c_t from the dataset D to produce $D' = D \setminus \{c_t\}$. For each example in D', with probability p, we replace that example's label with c_t and retrain the model for some iterations on this modified dataset.
4. **Class unlearning:** for each example labeled c_t we randomly assign another label from $C \setminus c_t$ and retrain for some iterations on this modified dataset before removing the c_t data.

Retraining: In the naive method of *retraining*, when the model owner receives a request for deletion from the individual represented by c_t, they remove all examples labeled c_t from D to produce a modified dataset $D' = D \setminus \{c_t\}$. Using the same training algorithm used to train the model, they continue training on D' for some iterations to produce updated model M' and inform the user that their data has been forgotten.

The attacker simply performs a MI attack on M' to try to generate an example of c_t. Our findings show that retraining is entirely insufficient to protect the individual, with inversion attacks successful even after a significant number of retraining iterations. Retraining maintains the quality of the model on all non-forgotten classes, but it requires a very substantial amount of training time to be even moderately effective, making it insufficiently efficient.

Neuron Removal: The slightly more savvy model owner may instead choose to remove the final layer neuron corresponding to c_t from the network entirely and release the new model M' that classifies $n-1$ classes. The owner may naively believe that if there is no output neuron for c_t then the individual is protected. The attacker can query M' to get an (n-1)-dimensional prediction vector P, calculate predictions, and can launch an MI attack on n-dimensional prediction vector $P \cup 0$, relating the added dimension to the forgotten class c_t.

Our findings show that a disconnected neuron does very little to protect the class from MI attacks. Unfortunately, in addition to removing the neuron, the model owner also has to perform a moderate amount of retraining on the modified dataset $D' = D \setminus \{c_t\}$ in order to ensure privacy. However, neuron removal does maintain the quality of the model on all non-forgotten classes. Also, even with the moderate amount of retraining described above, it is fairly efficient.

Scattered Unlearning: The intuition behind the scattered unlearning method is to dilute the unique aspects of the target class c_t with other random characteristics. As an example, consider trying to uniquely identify an animal from descriptions such as "the animal sometimes has 2 legs, sometimes has 4 legs, sometimes has 8 legs, and sometimes has no legs at all". The information learned is so diverse that no consensus can be gained.

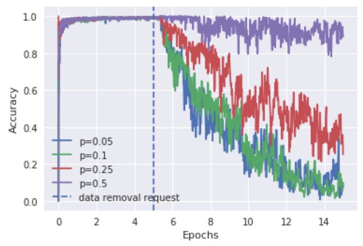

Figure 2: Effect of probability p on forgotten class accuracy

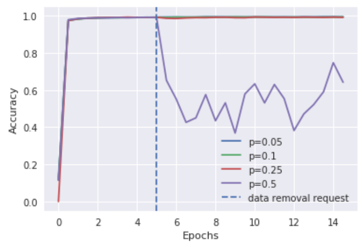

Figure 3: Effect of probability p on non-forgotten class accuracy

First, the model owner removes all instances of c_t from D to obtain the modified dataset D'. Second, for each example e in D', with some probability p, the model owner relabels e with c_t. The model M is retrained on the resulting dataset D' for some iterations to produce an updated model M' that is now defended by the scattered unlearning method.

Consider a scenario where the attacker performs an MI attack on M' to try to generate an example of c_t. As discussed below, our results show that a moderate amount of scattered unlearning makes MI attacks extremely difficult. One interesting note is that while inversion attacks are unsuccessful after a moderate amount of retraining, it takes a significant amount of retraining for the classification accuracy of c_t to reduce. This highlights the unsuitability of classification accuracy as a metric for how well c_t has been forgotten from the model.

In our experiments, we observed that the probability value p (with which the model owner may relabel examples with c_t) we choose has non-negligible effect on the efficacy of the defense mechanism. As seen in figure 2, the smaller the p value the faster the model loses the ability to correctly classify the forgotten class. However, larger p values cause more difficulty for inversion attacks with less retraining. However, as seen in figure 3, a sufficiently large p will cause the model to lose it's ability to correctly classify the rest of the data, rendering the model less suitable for classification. It's worth noting that when $p = 0$, scattered unlearning is indistinguishable from relearning which provides no defense against inversion attacks. We found $p = 0.1$ is a rule of thumb value to maintain model quality while providing robust defenses against inversion attacks.

Scattered unlearning maintains the quality of the model on non-forgotten classes. While modifying the dataset is a computationally expensive task, it is not significant enough to impact the efficiency of the method. Hence, this method is both effective and efficient.

Class Unlearning: In class unlearning, the model owner randomly relabels all examples in c_t with a new label from $C \setminus c_t$ to produce modified dataset D'. They then train M on D' for some iterations to produce updated model M'. Finally, they remove all examples of c_t from the dataset. Our results show class unlearning to be the most effective forgetting method for both classification accuracy and defense against inversion attacks while maintaining the quality of the model on non-forgotten classes. However, the data owner must hold a copy of the user's data during the retraining process - a restriction that could possibly have legal significance. Modifying the dataset is a computationally expensive task much like scattered unlearning, but class unlearning is both more efficient in terms of how much retraining is necessary and more effective in terms of resistance to inversion attacks.

5 Experimental Results

We trained a CNN on MNIST data [**?**] D for 5 epochs. Our network consists of 2 convolutional layers followed by 2 fully connected layers. We then separately applied each defense method to cause the models to forget the class *3* and trained each model for another 10 epochs (for scattered unlearning, we set $p = 0.1$). Using separate test sets, we evaluated model accuracy on $\{c_t\}$ and on $D \setminus \{c_t\}$ frequently during training. After each epoch we generated multiple MI attacks to evaluate how the inversion effectiveness changed as the defense was applied. All of our experiments were performed on an the Amazon SageMaker platform, using an EC2 P3.2xLarge instance.

We compare defense mechanisms along the following vectors: prediction accuracy on the class designated to be forgotten (for brevity we say forgotten class) as well as prediction accuracy on non-forgotten classes, resistance to MI attacks (evaluated based on observations). The ideal defense mechanism, according to this evaluation criteria, must have 0% prediction accuracy on forgotten class, must perform nearly as well as the original unmodified model on unforgotten classes, must be observationally resistant to MI attacks, and must be more efficient than other mechanisms considered. We found that class unlearning was the best along all these vectors.

5.1 Prediction Accuracy

Figure 4 shows the test accuracy results on the forgotten class during training. We see that for the models defended by class unlearning, the prediction accuracy quickly drops to 0% for the forgotten class (and as we show below, it is also resistant to MI attacks). Unfortunately, for the scattered unlearning defense mechanism, the prediction accuracy on forgotten class never goes to 0%, whereas the neuron removal model immediately loses any ability to classify the forgotten class. As we found, prediction accuracy is a necessary metric, but not sufficient to measure how well a class has been forgotten (e.g., neuron removal is excellent on this measure, but performs poorly against MI attacks). Additionally, we found that for all four defense mechanisms the prediction accuracy on the non-forgotten classes was unaffected. This shows that these methods do not damage the model's quality, a necessary property for a defense mechanism. Figure 5 shows the test accuracy on non-forgotten classes through training.

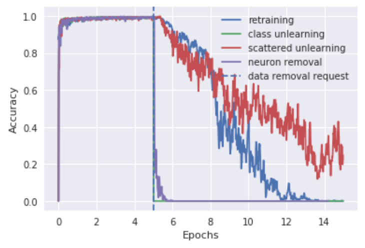

Figure 4: Test prediction accuracy on the forgotten class

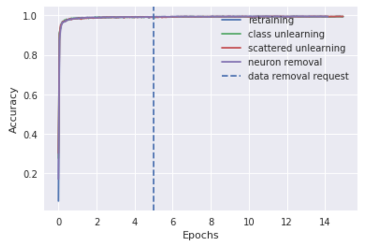

Figure 5: Test prediction accuracy on non-forgotten classes

5.2 Inversion Results

Retraining: As a baseline we considered retraining of the model with a modified data set where all examples labeled with to-be-forgotten label c_t removed. We found that retraining is an entirely insufficient method of protecting user data. The prediction accuracy of the forgotten class decreases very slowly, showing that this defense is insufficient in this regard as well. Additionally, MI attacks can reliably produce identifiable examples of c_t even after the prediction accuracy drops to 0% (see Figure 7). This highlights the fact that prediction accuracy cannot be used as a meaningful metric for data removal and showcases that inversions are an effective method to evaluate removal.

Neuron Removal: Our findings show that neuron removal alone isn't sufficient to protect data. A network that has learned to recognize c_t has also learned to reduce the likelihood that an example from c_t gets predicted to be another class. Hence, the values of the other final layer neurons (that are not responsible for predicting c_t) tend to be negative when given an example from c_t. If all other neurons have negative values, the "replaced neuron" with value 0 is the most probable class. As an example, a test forward pass of a digit *3* through such a network results in logit values of {-13.7, -2.31, -6.75, 0, -15.73, -2.44, -19.82, -3.87, -3.19, -4.7} which translate to probabilities $\{0, 0.08, 0, 0.79, 0, 0.07, 0, 0.02, 0.03, 0.01\}$. Not only does this allow the attacker to get a moderately successful prediction accuracy on c_t, but it allows them to very effectively perform MI attacks. Figure 6 shows an example of one such

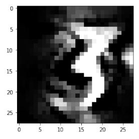

Figure 6: An inversion attack on M' with replaced neuron.

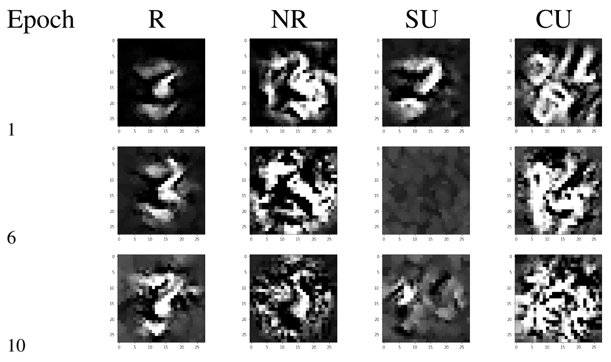

Figure 7: MI attack examples after 1, 6, and 10 epochs of retraining. Defense methods: *R*: retraining, *NR*: neuron removal, *SU*: scattered unlearning, *CU*: class unlearning

attack.

Scattered Unlearning: While the prediction accuracy for scattered unlearning method declines slower than the other methods, it is robust to MI attacks. The inverted images are recognizable for the first epoch but become unrecognizable soon after. Interestingly, after 6 epochs the prediction accuracy is around 45% while the inverted image is completely unrecognizable - highlighting again the fact that prediction accuracy on the forgotten class and resistance to inversion are not correlated. Further, since the model owner would not need to modify architecture of M or use the data from c_t to apply this method, this method may be the best option in certain contexts.

Class Unlearning: In terms of both prediction accuracy and robustness against inversion, class unlearning far outperforms other methods we considered in our experiments. Inversion attacks are completely unrecognizable after only 1 retraining epoch, while prediction accuracy on the forgotten class is 0% after a fraction of an epoch. A caveat of class unlearning is that it requires the model owner to keep a copy of the data that needs to be forgotten while they perform the retraining. While we do not know of any legal complications that could arise from holding this data during the forgetting process, we do not presume there are none.

Model Inversion Examples: We generated MI attack examples after each retraining epoch to test the effectiveness of the attacks. Some selected examples that were generated 1, 6, and 10 epochs after the RTBF request can be seen in figure 7. As can be seen, both the class and scattered unlearning methods vastly outperform the retraining and neuron removal methods on this measure.

6 Discussion

Metrics: We emphasize that MI attacks are not an exhaustive method of measuring privacy leaks from networks. Some models aren't as vulnerable to MI attacks, and MI attacks have a limited ability to give information about specific data examples themselves. However, finding a comprehensive method of evaluating all vulnerabilities of a model is very difficult, and to-date we have not found a good metric for this evaluation.

Black Box Attacks: One might argue that the impact of MI attacks can be limited by allowing clients to have only black box access to models. Unfortunately, model extraction attacks [**?**; **?**; **?**] have the ability to steal functionality of models even with only black box access. Further, the existence of these attacks mean that limiting clients to black box access is not sufficient to deter attackers who can steal models and then attack them under white box settings.

Stochastic Defenses: A straw defense we considered was a stochastic system where the model owner provides only black-box access to their system and modifies their model such that when it outputs the forgotten class, a post-processing filter randomly returns some other class. Otherwise, the model behaves as before. In keeping with the spirit of Kerckhoffs's principle, a defense that relies upon the attacker's ignorance cannot be considered secure. An attacker can make multiple queries using examples from the forgotten class to realize the presence of a stochastic defense, and this offers no more protection than a standard black-box model.

7 Conclusion

In this paper, we investigated how DNN models can be made to comply with the Right to be Forgotten regulation. In this scenario, model owners would not want to train a new model from scratch due to cost considerations, and there is a need to make the model forget target data without affecting its accuracy on rest of the data. We introduce several defense methods, namely, neuron removal, scattered unlearning, and class unlearning, that can be used to protect data privacy without incurring the significant cost of retraining a model. Further, we evaluated these models for model accuracy and weakness to model inversion attacks. We found that while all three methods are efficient and maintain model quality, the class unlearning method outperforms other methods on effectiveness against MI attacks as well as prediction accuracy. Given that this line of research into machine learning and law is new and a potentially very rich field, there remains a number of important problems to be resolved, including a mathematical formulation of the problem and solutions in the DNN setting that characterize defense mechanisms that are efficient, effective, and maintain model quality.