UnSplit: Data-Oblivious Model Inversion, Model Stealing, and Label Inference Attacks Against Split Learning

Ege Erdoğan[†], **Alptekin Küpçü**[†], **A. Ercüment Çiçek**[*‡]
[†] Department of Computer Engineering, Koç University
[*] Department of Computer Engineering, Bilkent University
[‡] Computational Biology Department, Carnegie Mellon University
{eerdogan17, akupcu}@ku.edu.tr, cicek@cs.bilkent.edu.tr

Abstract

Training deep neural networks requires large scale data, which often forces users to work in a distributed or outsourced setting, accompanied with privacy concerns. Split learning framework aims to address this concern by splitting up the model among the client and the server. The idea is that since the server does not have access to client's part of the model, the scheme supposedly provides privacy. We show that this is not true via two novel attacks. (1) We show that an honest-but-curious split learning server, equipped only with the knowledge of the client neural network architecture, can recover the input samples and also obtain a functionally similar model to the client model, without the client being able to detect the attack. (2) Furthermore, we show that if split learning is used naively to protect the training labels, the honest-but-curious server can infer the labels with perfect accuracy. We test our attacks using three benchmark datasets and investigate various properties of the overall system that affect the attacks' effectiveness. Our results show that plaintext split learning paradigm can pose serious security risks and provide no more than a false sense of security. [1]

1 Introduction

There has been a remarkable growth in the interest towards deep neural networks (DNNs) in the last decade, as they surpassed previous state-of-the-art machine learning models in many tasks, such as speech recognition [5] and natural language processing [3]. Aside from theoretical developments in DNN architectures and training methods, there has been two trends that still fuel this growth to date: increasing computing power, and availability of large data sets. Training a DNN with millions, even billions of parameters is an expensive task that requires significant computing power. It is also known that having access to large high-quality training data alone is generally enough to increase a model's performance [8]. Due to these reasons, distributed and outsourced approaches to model training that split the data storage and compute loads among multiple nodes have attracted attention in recent years.

Federated learning [2, 12, 13] and *split learning* (SplitNN) [7, 22, 23] are two distributed deep learning frameworks proposed to further the two trends described above. They achieve their goal by (i) enabling more efficient training of DNNs on devices with limited capabilities (e.g. smartphones, IoT devices), and (ii) making it possible for multiple data holders to collaboratively train a DNN without sharing their private data. However, various studies have shown that these techniques leak information [4, 6, 10, 16, 28, 29]. These distributed frameworks are promising a leap forward in

[1] Supplementary code can be found at `https://github.com/ege-erdogan/unsplit`.

Preprint. Under review.

fields like healthcare, which comes with stringent data privacy regulations. Thus, it is of critical importance to ensure that distributed neural network training is also privacy preserving.

The framework we focus in this paper, **split learning** or **SplitNN**, [7, 22, 23] allows one or more clients to jointly train a DNN by splitting the DNN so that the first few layers are computed at the client(s), and the rest of the layers are computed at a central server. Clients share the output of their final layer, rather than their private input data. The main security assumption behind SplitNN is that those outputs, called *smashed data*, do not leak information about the inputs. Compared to other similar frameworks, SplitNN stands out as being more efficient [23].

Our key contribution in this paper is UnSplit: A suite of two novel attacks against the SplitNN approach that effectively "unsplits" the split. Compared to previous similar attacks [10], our attacks work equally-well with the least amount of client-side knowledge needed by the attacker server. The two attacks can be summarized as follows:

- The first attack allows a SplitNN server to recover the inputs given to the model, while also obtaining a functionally similar model to the client model. Our only assumption for this attack is that the attacker knows the architecture of the client model. This leads to a very limited threat model, where the attack surface consists of the smashed data received from the client. We show that even in such a limited threat model, especially if the split layer is relatively early, the attacker can obtain pixel-perfect copies of the inputs, and a model that performs as well as the client model on unseen data.
- The second attack is a label inference attack that allows a honest-but-curious SplitNN server to infer the supposedly protected labels with perfect accuracy, under the assumption that the client part of the model has a depth of one. While this is a simplistic assumption, the effectiveness and potential consequences of the attack deems it worthy of discussion.

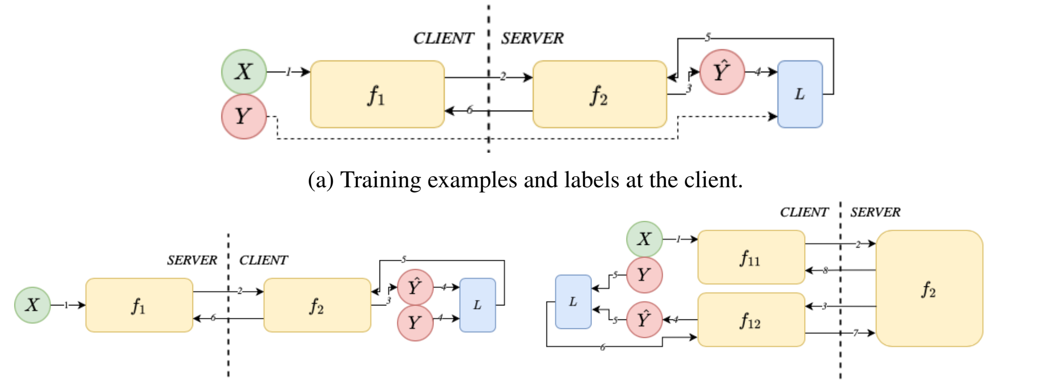
We should note that although we focus on the single-client setting in our experiments, our attacks are generalizable to multi-client setting as well. In both of our attacks, the server is an **honest-but-curious** attacker, meaning that the server acts according to the SplitNN protocol normally, but in the background performs the attack using the data it gathered throughout the protocol. Such attacks cannot be detected by a regular client, since everything in the original protocol works as expected. Thus, our adversary is very weak, requiring minimal assumptions, but the results of our attack are potentially devastating.

2 Background

SplitNN [7, 22, 23] is a distributed deep learning framework that enables multiple data holders and a central server to collaboratively train a DNN, without any of the data holders sharing their private data with other parties. Distributed methods such as SplitNN can provide substantial benefits in certain industries such as healthcare, where data holders (e.g. hospitals, clinics) are prohibited from sharing their data due to regulations such as HIPAA [1, 17].

The main idea behind SplitNN is to split and allot a DNN among multiple parties. Figure 1 displays potential setups of the SplitNN protocol. In its simplest setting (Figure 1a), SplitNN involves a single data holder (client) and a server. The client computes the first few layers of the DNN, and forwards the output, along with the target label, to the server. The server then resumes the computation with the remaining layers. This concludes a forward pass through the network, and then the server initiates the corresponding backward pass by computing the loss value. The server completes a backward pass through its part of the network, and sends the gradient values of its first layer back to the client. Finally, the client completes the backward pass, concluding a complete forward-backward pass through the network.

The above settings requires the client to share the training labels with the server. We can omit that requirement by further dividing the network into three parts, with the final part being computed at the client (Figure 1c). The training process is same as the previous setting with the addition of one more communication step. Since the loss value is computed at the client, the client does not need to share labels with the server. Alternatively, in another setting that does not require any data sharing, it can be the case that the server stores the training examples and the client stores the labels (Figure 1b).



(a) Training examples and labels at the client.

(b) Training examples and labels are split between the client and the server.

(c) Training examples and labels stored only at the client..

Figure 1: Three possible SplitNN setups. A client and a server train a model with the dataset containing examples X and labels Y, where $\hat{Y}$ stands for the model's predictions, and L for the loss function. Panel (a) displays the default setup in which the client possesses the training instances and labels, but transmits the labels to the client for the server to compute the loss function. Panel (b) shows a similar setup, but with the training instances at the server, and the labels at the client. Extending from the setup in Panel (a), Panel (c) displays the setup in which the client does not have to share the labels with the server. The last layers of the model, as well as the loss function, are computed by the client. The numbers on the edges denote the steps of computation in order.

To accommodate multiple clients around the settings described above, a turn-based training procedure is employed in which the clients take turns training with the server. Before a client starts its turn, it updates is weights with that of the most recently trained client. This can be achieved either through a central server, or in a peer-to-peer basis between the clients. In the present work, we focus on the two-part setup with a single client and a server, although the attacks can be employed in a setting with multiple clients as well.

As described earlier, the ability of SplitNN to maintain clients' input privacy relies on the assumption that original inputs cannot be recovered from the smashed data. If such an attack is possible under a reasonable threat model, then additional security measures would be required to fulfill the privacy guarantees of SplitNN.

3 Related Work

A model inversion (MI) attack [4, 6, 10, 16, 25, 28, 29] involves an adversary trying to recover the input fed to a machine learning model, given access to its output. An early example of a model inversion attack was an attack proposed by Fredrikson et al. [6] targeting a linear regression model used to adjust personalized Warfarin doses for patients. Given the machine learning model and some demographic information about a patient, the attack was able to predict the patient's genetic markers used as inputs to the model.

Table 1 summarizes the threat models of various MI attacks. Notice that it is assumed for most attacks that the attacker can send queries to the target model. However, that is not possible for an attack performed by a SplitNN server since the clients control the inputs given to the model.

In the first in-depth security analysis of SplitNN, Pasquini et al. [18] showed that it is possible for an honest-but-curious server to obtain the clients' private training data during the training phase. Their attack relies on the server's ability to manipulate the client during the training process by propagating back loss values that are unrelated to the original task, but aid the server in its pursuit. The results of the attack demonstrate that the SplitNN protocol is inherently insecure. However, this attack cannot steal the client model.

The attack, named the Feature-Space Hijacking Attack (FSHA), assumes an attacker that has access to a data set X_{pub} that follows a similar distribution with that of the client's training data X_{priv}. Briefly, the attacker trains an autoencoder on X_{pub} and directs the client towards outputting values belonging

Table 1: Threat models for various model inversion attacks. All attacks aim to reconstruct inputs given to the target model. UnSplit aims to steal the model as well. Attacks that directly target SplitNN are marked with an asterisk (*) and are at the bottom half of the table.

Attack	Adversary Knowledge / Capabilities				
	Target Model		Training Data		Send
	Structure	Parameters	Values	Distribution	Queries
Zhang et al. [27]	×	×	-	-	×
Salem et al. [21]	-	-	-	-	×
Zhu et al. [29]	×	×	-	-	×
Fredrikson et al. (White-box) [6]	×	×	-	-	×
Fredrikson et al. (Black-box) [6]	-	-	-	-	×
He et al. (White-box) [10]*	×	×	-	-	-
He et al. (Black-box) [10]*	-	-	×	×	×
He et al. (Query-free) [10]*	×	-	×	×	-
Pasquini et al. [18]*	-	-	-	×	-
UnSplit*	×	-	-	-	-

to the same latent space as the encoder part of the autoencoder. Since the decoder essentially knows how to invert the values belonging to that latent space, it is able to invert values received from the client, and obtain the original inputs. The main difference of FSHA and UnSplit is that we do not have any assumptions about the attacker's knowledge of a public data set related to the original task. Without such a data set, FSHA becomes infeasible, since the attacker cannot train the autoencoder network. Again, FSHA cannot steal the client model.

In a different set of MI attacks targeting collaborative inference systems similar to SplitNN, He et al. [10] showed under three different threat models that it is possible to recover the input fed to a DNN with different degrees of accuracy. They considered *white-box* scenarios, where the adversary already knows the parameters of the DNN, *black-box* scenarios, where the adversary does not know the parameters but can query the DNN, and *query-free* scenarios, where the adversary neither knows the weights of nor can send queries to the DNN, but knows about the underlying dataset. Under the white-box setting, since the adversary can perform gradient descent on the initial part of the network, the attack corresponds to an optimization problem. The adversary starts with a random input, and updates it with gradient descent until the output becomes close enough to the actual output. In the black-box setting, by querying the original model, the adversary trains an inverse-network to learn a mapping similar to the inverse of the original network, and uses that inverse-network to obtain the inputs from the intermediate values. Finally, in the query-free setting, utilizing either the original training dataset or a different dataset following a similar distribution, the adversary trains its own network that performs similarly to the original network. The attack then follows a similar structure with the white-box setting described earlier, with the exception that now the adversary works with its clone of the first part of the model. It should be noted that the effectiveness of the black-box and the query-free attacks depends on the adversary's prior knowledge of the original learning task. Note that while this study is not concerned with SplitNN in particular, it is still applicable and we include it in our benchmarks.

In a related but inapplicable setting, targeting federated learning, Zhu et al. [29] shows that an honest-but-curious server could recover an input by trying to find a value such that the resulting gradient values match those shared by the data holders. This implies that even when the forward and backward passes are performed on the client side, sharing the gradient values can leak information.

4 Method

4.1 Threat Models

For the **model inversion and stealing attack**, we consider a client and a server running the SplitNN protocol, where for simplicity a DNN F is partitioned into two parameterized functions f_1 and f_2

such that $F(\theta, x) = f_2(\theta_2, f_1(\theta_1, x))$, although the attack can be extended without loss of generality to the setup in which the client also controls the final layers of the network (Figure 1c).

We assume an attacker that knows the model architecture, but not the parameters, of f_1. The attacker does not have access to any specific data, and strictly follows the SplitNN protocol. This means that the attacker cannot query the client network, and does not send training updates other than the one required for the original learning task. It is worth noting that whether the model terminates at the client (Figure 1c) or the server (Figure 1a) is of no importance to the attacker under this threat model, since everything the attacker needs is the intermediate activation values she received from the client. Thus, we model an ***honest-but-curious*** attacker, which is a much weaker form compared to a powerful malicious attacker.

The attacker's goals are to recover any input given to the network F, and obtain a functionally similar clone $\tilde{f_1}$ of the client network f_1. Functional similarity in this context concerns the models' performance (e.g. classification accuracy) on unseen data, such as the test set corresponding to the training set. Within this threat model, it is impossible for the clients to distinguish a server launching the attack from one following the protocol.

It is important to note that this is a realistic scenario for SplitNN. Consider a researcher training a DNN, aggregating data from multiple healthcare providers. In this scenario, the researcher acts as the SplitNN server, and the healthcare providers act as clients. It is also realistic to expect that the researcher knows the reference architecture f_1 that the healthcare providers employ, but maybe not the associated parameters. A similar scenario can involve an application developer as the SplitNN server, and the user devices (e.g., smartphones) as the clients.

For the **label inference attack** (Figure 1b), the assumptions made for the model inversion and stealing attack remain valid. Those assumptions imply that the attacker knows how many discrete labels there are. We further assume that training updates are calculated with *stochastic* gradient descent, and that the client model has a depth of one. This assumption is feasible because it is reasonable to expect that a protocol such as SplitNN to be used in a setup with minimal cost for the clients, while "protecting" their data. The severity of the attack's possible consequences deems it worthy of discussion, and highlights the importance of warning against such misguided use.

4.2 Model Inversion & Stealing

Without any data similar to the training data or the ability to query the client network, the attacker's task is essentially a search over the space consisting of all possible input values and client network's parameters. We model the problem as an optimization problem: the attacker tries to find parameters $\tilde{\theta_1}$ and input $\tilde{x}$ to minimize the difference between $\tilde{f_1}(\tilde{\theta_1}, \tilde{x})$ and $f_1(\theta_1, x)$ (Equations 1 and 2).

The optimization problem described above can be solved with gradient-based methods. However, we have observed in our experiments that performing the updates on the input and parameters simultaneously, in a single gradient update, often does not yield favorable results. Instead, we adopt a "coordinate gradient descent" [24] approach. A coordinate descent involves keeping a subset of the parameters fixed while updating another subset. The choice of which subset to update can be made randomly, or the parameters can be updated in a round-robin manner, as in UnSplit.

In UnSplit, we partition the target values into two sets, following their logical separation: the input values $\tilde{x}$ and the parameter values $\tilde{\theta_1}$. Given some intermediate output $f_1(x)$ of the client, the attacker first performs gradient descent updates on the estimated input values $\tilde{x}$, keeping $\tilde{\theta_1}$ fixed, and then repeats the same process by keeping $\tilde{x}$ constant and updating $\tilde{\theta_1}$. Algorithm 1 in the supplementary material summarizes the attack.

The attack can be modified to obtain more accurate results by tuning various parameters on different levels. First, the attacker can set the number of gradient descent steps separately for both $\tilde{x}$ and $\tilde{\theta_1}$, as well as the total number of rounds. Furthermore, different optimization algorithms and objective functions can be used to update the $\tilde{x}$ and $\tilde{\theta_1}$ values separately. Finally, even though we describe the attack using the binary partitioning mentioned above, the attacker can be thought of as having control over the partitioning of the search space as well, either dividing into more sub-spaces (e.g. by separating each layer's parameters), or merging into a single space.

To begin the *model inversion and stealing attack*, the server randomly initializes a model that has the same architecture with the client model. Then, the attacker defines two objective functions: one for the input update steps, and one for model update steps. We minimize the mean squared error (MSE) for both updates. Note that this is independent of the loss function used for the actual training task. Furthermore, since we are working in the image domain (see Section 5), we also add a Total Variation [20] term to be minimized, following from the work in [10]. Total Variation is a measure of the noise present in an image, and minimizing it results in smoother images. It is defined for an image x as

$$\text{TV}(x) = \sum_{i,j} \sqrt{|x_{i+1,j} - x_{i,j}|^2 + |x_{i,j+1} - x_{i,j}|^2},$$

where i and j denote the pixel indices.

In the end, we can summarize the attacker's task with Equations 1 and 2. The coefficient λ can be set to modify how much the Total Variation term affects the loss function.

$$\tilde{x}^* = argmin_{\tilde{x}}\,\text{MSE}(\tilde{f_1}(\tilde{\theta_1}, \tilde{x}), f_1(\theta_1, x)) + \lambda\text{TV}(\tilde{x}) \tag{1}$$

$$\tilde{\theta_1}^* = argmin_{\tilde{\theta_1}}\,\text{MSE}(\tilde{f_1}(\tilde{\theta_1}, \tilde{x}), f_1(\theta_1, x)) \tag{2}$$

4.3 Label Inference

Before launching the label inference attack, the attacker receives the gradient values from the client layer resulting from a single training example during backpropagation. The attacker also knows the input given to the client model as part of the protocol. Figures 1b and 1c are potential SplitNN setups in which the server can perform label inference.

To launch the attack, the attacker randomly initializes a model $\tilde{f_2}$ that has the same architecture with the client model f_2. The attacker then computes the gradient values resulting from backpropagation on $\tilde{f_2}$ for each possible label. The label value that produces the closest gradient values to the gradient values received from the client is output as the predicted label. The attacker can then train its clone model with the predicted labels.

To summarize, as displayed by Equation 3 below and Algorithm 2 in the supplementary material, the attacker finds the label $\tilde{y}^*$ that minimizes the distance between the gradients computed from the clone model and those received from the client.

$$\tilde{y}^* = argmin_{\tilde{y}}\; MSE(\frac{\partial L(f_2(f_1(x)), y)}{\partial \theta_2}, \frac{\partial L(\tilde{f_2}(f_1(x)), \tilde{y})}{\partial \tilde{\theta_2}}) \tag{3}$$

5 Experimental Results

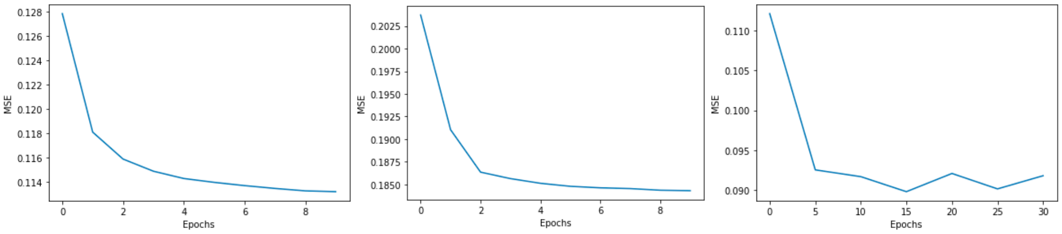
5.1 Experimental Setup

We perform the experiments with three widely-used image classification benchmark datasets, with 10 classes each: MNIST [15], Fashion-MNIST [26], and CIFAR10 [14]. The datasets are well-known datasets that do not contain any personally identifiable information or offensive content.

We make use of the models used in [10] consisting of several convolutional and fully connected layers, with ReLU and sigmoid activations. To keep the discussion of the attack independent of the DNN specifics, we describe the detailed architectures in the Supplementary Material.

We train the original client model using the entire training partition of the datasets, and test the clone model's performance using the test partition of the respective datasets. We pick our sample sets from the test sets of the respective datasets. Each sample set contains one instance of each class, for a total of 10 examples. We perform no post-processing on the estimated inputs. For the sake of brevity, and taking into account that late splits defy the efficient outsourcing purpose of SplitNN, we conduct the experiments for the first six possible layer splits for MNIST and Fashion-MNIST, and the first eight possible layer splits for CIFAR10.

For the model inversion loss function (Equation 1), we set the TV coefficient λ to be 0.1 for the first three split layers, and 1 for the rest. We use the Adam optimizer [11] with a learning rate of 0.001 to perform the gradient descent updates.



(a) MNIST (b) Fashion-MNIST (c) CIFAR10

Figure 2: MSE values of the estimated inputs after each training epoch, averaged over possible split layers. It can be observed that the estimates become more accurate as the client model is trained.

Table 2: Mean squared error (MSE) values for the original and estimated inputs obtained when the attack is performed before and after the training phase, and the clone model's classification accuracy on the test sets when the attack is performed after the training phase. The missing values corresponds to the splits we did not run experiments for.

	MNIST			F-MNIST			CIFAR10		
Split Depth	MSE Before Train	MSE After Train	Test Acc. % (ref: 98)	MSE Before Train	MSE After Train	Test Acc. % (ref: 88)	MSE Before Train	MSE After Train	Test Acc. % (ref: 71)
1	0.077	0.054	97.45	0.146	0.127	86.11	0.039	0.062	58.03
2	0.084	0.056	95.69	0.153	0.144	84.34	0.065	0.071	54.02
3	0.084	0.060	93.75	0.157	0.147	81.24	0.098	0.052	55.15
4	0.099	0.119	76.27	0.168	0.084	66.17	0.099	0.068	43.69
5	0.109	0.096	65.27	0.180	0.147	11.54	0.109	0.092	46.75
6	0.108	0.137	63.3	0.177	0.131	16.12	0.089	0.083	18.54
7	-	-	-	-	-	-	0.087	0.088	16.30
8	-	-	-	-	-	-	0.092	0.070	22.12

We implement the attack in Python (v3.7) using the PyTorch library (v1.7.1) [19]. The time to invert a single input ranged between one and five minutes using a personal computer (2.9 GHz Intel i7 CPUs).

5.2 Results

Table 4 displays the estimated inputs obtained from the model inversion and stealing attack. More detailed results are displayed in Table 6 found in the Supplementary Material. Table 2 displays the MSE values between the original and estimated inputs, as well as the classification accuracy of the clone model. When the client model is trained, the attacker estimates inputs with reconstruction errors of 0.087, 0.13, 0.0733 on average for MNIST, Fashion-MNIST, and CIFAR10 datasets, respectively. Against an untrained client model, the error values increase to 0.094, 0.164, and 0.085, implying that a trained model is more vulnerable to an attack compared to an untrained model. Furthermore, especially for early splits, the clone model performs very close to the original model on previously unseen data. Averaging over the first three splits, the clone model achieves a test classification accuracy of 95.63% for MNIST, 83.90% for Fashion-MNIST, and 55.73% for CIFAR10.

Label Inference. We observe that under the assumption of a client computing only the last layer, aiming to hide the labels from the server while delegating as much work as possible, the attacker can infer the labels with perfect accuracy (Table 3). After successfully inferring the labels, the attacker can then train its clone model and obtain a model that performs as well as the client model. This implies that if the client part of the network intended to protect the training labels from the server has a depth of one layer, it simply does not achieve its

Table 3: Label inference accuracy for the three benchmark datasets when the client part has depths of one and two.

Client	Label Inference Accuracy		
Depth	MNIST	F-MNIST	CIFAR10
1	100%	100%	100%
2	9.1%	10.2%	8.1%

purpose. However, given that each of our benchmark datasets has 10 label values, the values displayed in Table 3 indicate that the attacker's guesses become no better than random guesses when the client model has a depth of two layers.

Effect of training state. It can be observed in Table 2 that the quality of the estimated inputs is higher when the attack is performed after the training phase rather than before. Furthermore, Figure 2 details how the MSE values between the original and the estimated inputs progress through the model training epochs. It is again visible that the estimates become more accurate as the model is trained. This is expected since a trained model's output contains more information about the inputs compared to a random, untrained model. However, it is misleading to think that an untrained model is not vulnerable to the attack. As the results displayed Table 6 (in the Supplementary Material) demonstrate, an untrained model is vulnerable as well.

Comparing with other attacks. Figure 3 and Table 4 compare the results obtained from UnSplit with the attacks in [10]. To compare the attacks under similar threat models, we assume that the attacker does not have access to any specific dataset in any of the scenarios. The estimates generated by the white-box attack are on average 38% more accurate than those generated by UnSplit. This is expected since the adversary knows the model parameters in the white-box scenario, which is a significant, though unrealistic, advantage. On the other hand, UnSplit results in more accurate estimates by an average of 32% compared to the query-free attack, which has a more similar adversarial capability. Finally, it can be seen from Table 5 that UnSplit performs comparable to the black-box attack in [10] when the attacker has a very limited query budget. The required query values are obtained by averaging 30 runs of the He et al. [10] attack with 500 images each. However, taking into account the details of the default SplitNN setup (Figure 1a), it is not possible for the server to send queries to the client without violating the protocol. Therefore, black-box attack is not an honest-but-curious attacker model.

Table 4: Comparison of estimated inputs obtained from the white-box and query-free scenarios in [10] and UnSplit for the same split depth of three for each of our benchmark datasets.

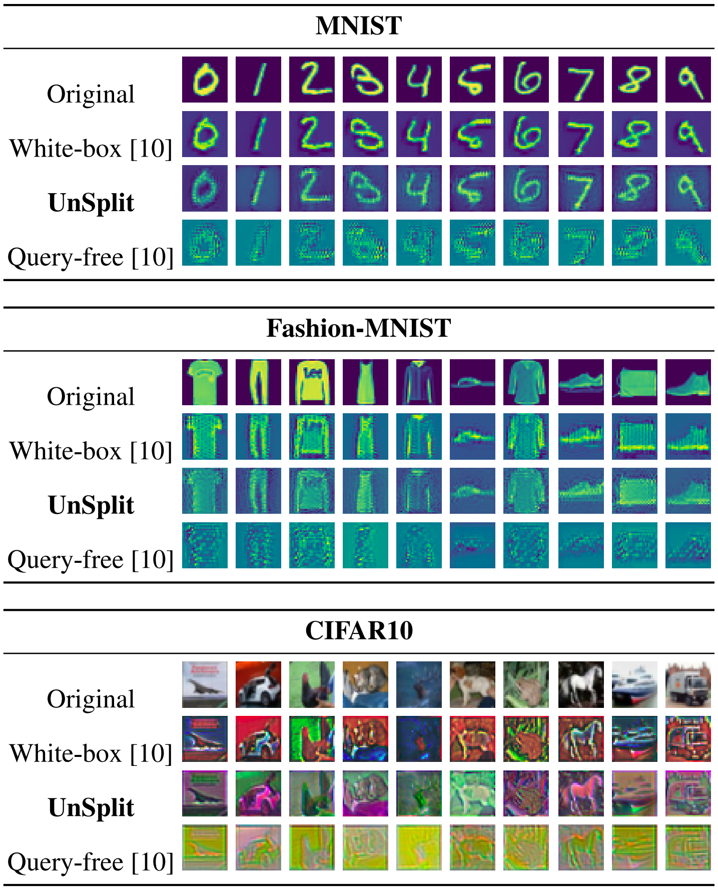
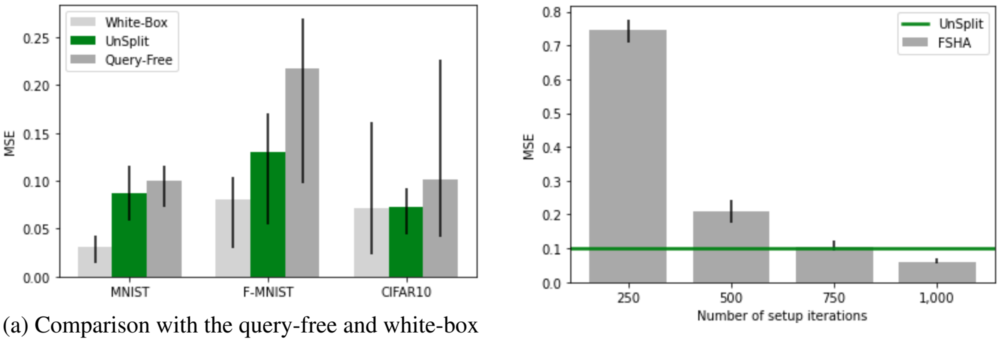

MNIST
Original
White-box [10]
UnSplit
Query-free [10]
Fashion-MNIST
Original
White-box [10]
UnSplit
Query-free [10]
CIFAR10
Original
White-box [10]
UnSplit
Query-free [10]

Figure 3b displays the results of a comparison between UnSplit and FSHA [18] on the MNIST dataset. This comparison was performed with the ResNet [9] architecture unlike the rest of our experiments. It can be observed that UnSplit performs comparably to FSHA until the FSHA adversary performs around 1,000 setup iterations. Note that the adversary in FSHA is stronger, with access to a dataset



(a) Comparison with the query-free and white-box attacks in [10].

(b) Comparison with FSHA.

Figure 3: Comparison of UnSplit with the attacks described in [10] for white box and query-free scenario, and FSHA [18]. The figures displays the MSE values between the estimated inputs and the original values averaged over possible split layers. The vertical lines represent the upper and lower quartiles of the values, while the bars correspond to the means.

that follows a similar distribution to the training set. The FSHA attack becomes infeasible without such a dataset.

6 Limitations

Our main assumption for both attacks is that the attacker knows the model architecture used at the client's part. For the label inversion attack, we further assumed that the client only performs the last layer, and that the training is done with *stochastic* gradient descent as opposed to *batch* gradient descent. We believe the consequences of the attack still make it worth presenting. Finally, we test our attacks only on image data. Thus, we leave attacks with less attacker knowledge regarding both model inversion and label inversion, and more client layers and batch gradient descent regarding label inversion as future work.

7 Conclusion

Our attacks demonstrate that with the knowledge of the client's DNN architecture alone, it is possible for a honest-but-curious SplitNN server to obtain the inputs given to the model, and a model that performs similarly to the original client model. Furthermore, under the assumption that the final client split has a depth of one, the server can infer the labels with perfect accuracy. These attacks considered together effectively "unsplit" the split learning approach. Thus, it is of critical importance to warn against such allegedly secure yet blatantly insecure uses of the SplitNN protocol.

Expectedly for the model inversion and stealing attack, its effectiveness decreases as the split layer becomes deeper. This is not surprising since the earlier layers of a DNN contain more information about the inputs. This introduces a performance/security trade-off for the clients. If the data being fed into the DNN is sensitive (e.g. patient data in a clinic), then the data holders can increase the security of the protocol by essentially spending more computing power.

Table 5: Number of black box queries needed for the black box attack scenario in [10] to obtain the same MSE values as UnSplit.

Split	Number of Black Box Queries		
Depth	MNIST	F-MNIST	CIFAR10
1	3.61	2.11	7.04
2	7.81	3.89	8.42
3	9.50	9.66	9.48
4	7.05	9.50	8.64
5	9.06	9.51	9.82
6	9.38	9.50	9.34

However, even though expanding more computing resources by way of computing more layers *increases* the security of the protocol, it does not *guarantee* it. Additional mechanisms such as homomorphic encryption are required to provide provable security guarantees. The possibility of our attack under a limited threat model exposes the inherent insecurity of vanilla SplitNN, and highlights the importance of such additional measures to yield the protocol secure.

Acknowledgements

We acknowledge support from TÜBİTAK, the Scientific and Technological Research Council of Turkey, under project number 119E088.

References

[1] G. J. Annas. HIPAA Regulations — A New Era of Medical-Record Privacy? *New England Journal of Medicine*, 348(15):1486–1490, Apr. 2003.

[2] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečný, S. Mazzocchi, H. B. McMahan, T. Van Overveldt, D. Petrou, D. Ramage, and J. Roselander. Towards Federated Learning at Scale: System Design. *arXiv:1902.01046 [cs, stat]*, Mar. 2019. arXiv: 1902.01046.

[3] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020. arXiv: 2005.14165.

[4] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. "You Might Also Like:" Privacy Risks of Collaborative Filtering. In *2011 IEEE Symposium on Security and Privacy*, pages 231–246, Oakland, CA, USA, May 2011. IEEE.

[5] C.-C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, E. Gonina, N. Jaitly, B. Li, J. Chorowski, and M. Bacchiani. State-of-the-art Speech Recognition With Sequence-to-Sequence Models. *arXiv:1712.01769 [cs, eess, stat]*, Feb. 2018. arXiv: 1712.01769.

[6] M. Fredrikson, S. Jha, and T. Ristenpart. Model Inversion Attacks that Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security - CCS '15*, pages 1322–1333, Denver, Colorado, USA, 2015. ACM Press.

[7] O. Gupta and R. Raskar. Distributed learning of deep neural network over multiple agents. *arXiv:1810.06060 [cs, stat]*, Oct. 2018. arXiv: 1810.06060.

[8] A. Halevy, P. Norvig, and F. Pereira. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24(2):8–12, Mar. 2009.

[9] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[10] Z. He, T. Zhang, and R. B. Lee. Model inversion attacks against collaborative inference. In *Proceedings of the 35th Annual Computer Security Applications Conference*, pages 148–162, San Juan Puerto Rico, Dec. 2019. ACM.

[11] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]*, Jan. 2017. arXiv: 1412.6980.

[12] J. Konečný, H. B. McMahan, D. Ramage, and P. Richtárik. Federated Optimization: Distributed Machine Learning for On-Device Intelligence. *arXiv:1610.02527 [cs]*, Oct. 2016. arXiv: 1610.02527.

[13] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated Learning: Strategies for Improving Communication Efficiency. *arXiv:1610.05492 [cs]*, Oct. 2017. arXiv: 1610.05492.

[14] A. Krizhevsky. Learning multiple layers of features from tiny images. 2009.

[15] Y. LeCun, C. Cortes, and C. Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[16] O. Li, J. Sun, X. Yang, W. Gao, H. Zhang, J. Xie, V. Smith, and C. Wang. Label Leakage and Protection in Two-party Split Learning. *arXiv:2102.08504 [cs]*, Feb. 2021. arXiv: 2102.08504.

[17] R. T. Mercuri. The HIPAA-potamus in health care data security. *Communications of the ACM*, 47(7):25–28, July 2004.

[18] D. Pasquini, G. Ateniese, and M. Bernaschi. Unleashing the Tiger: Inference Attacks on Split Learning. *arXiv:2012.02670 [cs]*, Jan. 2021. arXiv: 2012.02670.

[19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[20] L. I. Rudin, S. Osher, and E. Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 60(1-4):259–268, Nov. 1992.

[21] A. Salem, A. Bhattacharya, M. Backes, M. Fritz, and Y. Zhang. Updates-Leak: Data Set Inference and Reconstruction Attacks in Online Learning. *arXiv:1904.01067 [cs, stat]*, Nov. 2019. arXiv: 1904.01067.

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

[23] P. Vepakomma, T. Swedish, R. Raskar, O. Gupta, and A. Dubey. No Peek: A Survey of private distributed deep learning. *arXiv:1812.03288 [cs, stat]*, Dec. 2018. arXiv: 1812.03288.

[24] S. J. Wright. Coordinate Descent Algorithms. *arXiv:1502.04759 [math]*, Feb. 2015. arXiv: 1502.04759.

[25] X. Wu, M. Fredrikson, S. Jha, and J. F. Naughton. A Methodology for Formalizing Model-Inversion Attacks. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*, pages 355–370, Lisbon, June 2016. IEEE.

[26] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. 2017.

[27] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song. The Secret Revealer: Generative Model-Inversion Attacks Against Deep Neural Networks. *arXiv:1911.07135 [cs, stat]*, Apr. 2020. arXiv: 1911.07135.

[28] B. Zhao, K. R. Mopuri, and H. Bilen. iDLG: Improved Deep Leakage from Gradients. *arXiv:2001.02610 [cs, stat]*, Jan. 2020. arXiv: 2001.02610.

[29] L. Zhu, Z. Liu, and S. Han. Deep Leakage from Gradients. *arXiv:1906.08935 [cs, stat]*, Dec. 2019. arXiv: 1906.08935.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Section 6.
   (c) Did you discuss any potential negative societal impacts of your work? [Yes]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] No theoretical results were included.
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Reference URL omitted to comply with the double-blind review procedure.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

   (a) If your work uses existing assets, did you cite the creators? [Yes]

   (b) Did you mention the license of the assets? [Yes]

   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Reference URL omitted to comply with the double-blind review procedure.

   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]

   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes]

5. If you used crowdsourcing or conducted research with human subjects...

   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] We neither used crowdsourcing nor conducted research with human subjects.

   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Model Architectures

The following appendices explain the DNN architectures we used in our experiments.

A.1 MNIST and Fashion-MNIST

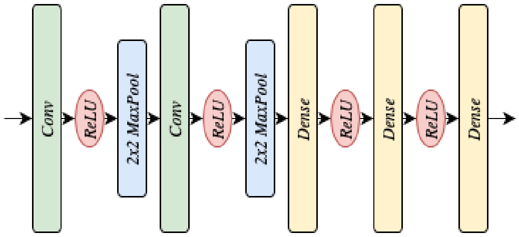


Figure 4: The DNN architecture we used in our experiments for the MNIST and Fashion-MNIST datasets.

The DNN we used for the MNIST and Fashion-MNIST datasets (Figure 4) consists of two convolutional layers, the first with 8 and the second with 16 output channels. Each convolution is followed by a 2x2 max pooling and and ReLU activations. Finally, there are three fully-connected layers, again with ReLU activations, and the softmax function is applied in the end to obtain the probability values for labels.

A.2 CIFAR10

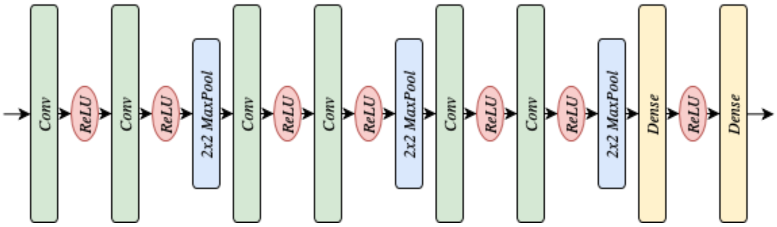


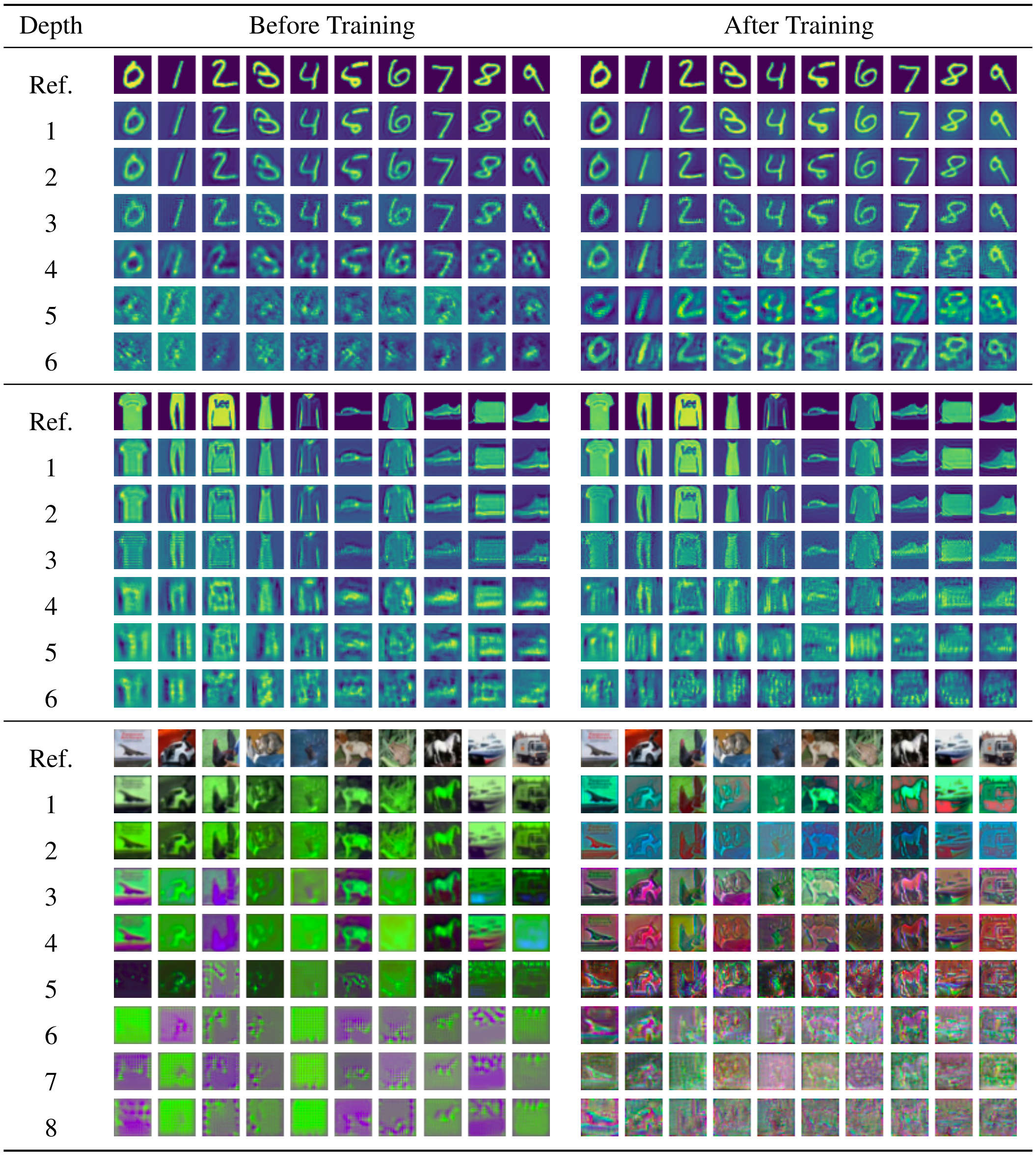
Figure 5: The DNN architecture we used in our experiments for the CIFAR10 dataset.

The DNN used for CIFAR10 (Figure 5) follows a similar but deeper architecture compared to the model used for MNIST and Fashion-MNIST. Two convolutional layers, each with 64 output channels and ReLU activations are applied, followed by a 2x2 max pooling layer. The same structure (two convolutions and a max pooling) is then repeated two more times, but with the convolutional layers outputting 128 channels. The DNN again ends with two fully-connected layers with sigmoid activations.

B Experimental Results

Table 6 displays the estimated inputs for various possible split layers for each of our benchmark datasets.

Table 6: Estimated inputs before and after the training phase for different split layers and the MNIST, F-MNIST, and CIFAR10 datasets. The first rows (Ref.) display the actual inputs, and the following rows display the estimates for different split depths as denoted in the Depth column.



C Algorithms

```
Algorithm 1: UnSplit: Model Inversion & Stealing
Result: $\tilde{x}^*$ and $\tilde{\theta_1}^*$
L: objective function
$x$: training example
$f_1$: client model
$f_2$: server model
$\tilde{f_1}$: randomly initialized copy of the client model
Repeat until convergence:
    $\tilde{x}^* = argmin_{\tilde{x}} \, \mathrm{L}(\tilde{f_1}(\tilde{\theta_1}, \tilde{x}), f_1(\theta_1, x)) + \lambda \mathrm{TV}(\tilde{x})$
    $\tilde{\theta_1}^* = argmin_{\tilde{\theta_1}} \, \mathrm{L}(\tilde{f_1}(\tilde{\theta_1}, \tilde{x}), f_1(\theta_1, x))$
```

```
Algorithm 2: UnSplit: Label Inference
Result: $\tilde{y}^*$
$L$: objective function
$(x, y)$: training examples and labels
$f_1$: server model
$f_2$: client model
$\tilde{f_2}$: randomly initialized copy of the client model
$h = \frac{\partial L(f_2(f_1(x)),y)}{\partial \theta_2}$
$\tilde{y}^* = argmin_{\tilde{y}} \, MSE(h, \frac{\partial L(\tilde{f}_2(f_1(x)),\tilde{y})}{\partial \tilde{\theta_2}})$
```