Patch-based Privacy Preserving Neural Network for Vision Tasks

Mitsuhiro Mabuchi
Toyota Motor Corporation
Tokyo, Japan
mitsuhiro_mabuchi@mail.toyota.co.jp

Tetsuya Ishikawa
Woven Core, Inc.
Tokyo, Japan
tetsuya.ishikawa@woven-planet.global

Abstract

As machine learning technology is increasingly adopted into a variety of application domains, the potential risks of data leakage are becoming more serious in certain cases where the data contains highly sensitive information. While some privacy-preserving learning mechanisms for image data, such as SplitNN, enable the training of models without sharing private data on a central server, there exists a trade-off between security and computational cost to a client device. We propose a new mechanism to achieve higher level security and lower computational cost on a client device while maintaining model performance. Our approach, called Patch SplitNN, is based on SplitNN architecture that divides a CNN into two networks, called upper and lower. The difference from that previous work is to input individual image patches into multiple upper models, before concatenating their outputs before the lower model. For further improvement of the upper model training, we introduce an additional network and a loss function into the training process. We demonstrate our Patch SplitNN can classify images as accurately as a ResNet18 on various image classification datasets (CIFAR-10, CIFAR-100, and PCam) under multiple conditions (e.g. patching patterns, dropping patches).

1. Introduction

Preserving the privacy of individuals is a key challenge when trying to solve many human-related problems. Distributed learning mechanisms as represented by Federated Learning [8, 9, 25] or Split Learning [15, 22] can train machine learning models while retaining training data outside a central training server, on a client, edge device, or edge server. The central training server receives model parameters or gradient values instead of original data in order to train a model. However, in these mechanisms, further analysing or labeling original data is one of the issues, especially when a model accuracy is lower than the target.

Once gathering data on a central server, data must be protected from leakage. Encryption is a standard approach to reduce the risk of leakage. However, even if data is encrypted while storing, decrypted data is used in training and it might be a possible attack surface.

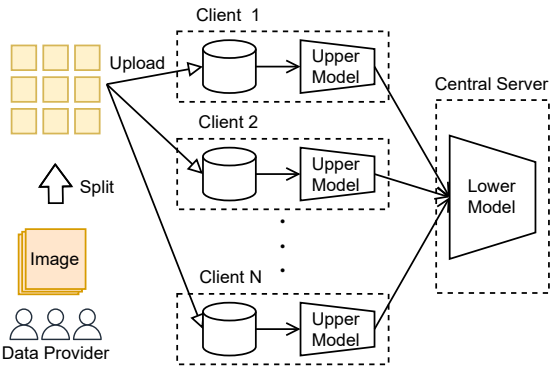

Figure 1. Overview of Patch SplitNN architecture.

Homomorphic encryption [1, 2, 3, 4, 6, 11] enables to train ML models without decrypting the input data. However, it cannot be a reasonable solution to train a deep neural network due to its extremely high computational load. Another option is to remove privacy-sensitive information from collected data, however, this approach is not suitable for some computer vision problems. For example, faces are highly privacy-sensitive information, but we cannot remove faces from data for training a face recognition model.

To achieve both privacy-preservation and computational efficiency while maintaining the same performance as a state-of-the-art computer vision model, we propose Patch Split Neural Network (hereinafter abbreviated as Patch SplitNN) architecture based on Split Learning (Figure 1). Patch SplitNN can protect privacy by preventing information leakage. Even if a patch image is leaked, it would not be considered a privacy leak when the leaked patch is small enough, and the amount of leaked information (number of bits) is much smaller than the original image.

Patch SplitNN consists of three parts. The first part, inspired by Vision Transformer [5] and ConvMixer [20], is to

split an image into patches which are small enough to make privacy indistinguishable. These patches are uploaded securely by using a common secure connection protocol such as *HTTS* or *TLS* and stored in separate storage, and each storage should be accessed by authorised servers or people. Since generating patches is not encryption but has a similar anonymizing effect, the computational effort is equivalent to training a base model.

The second part, that is inspired by SplitNN, is to divide CNN into two parts: an upper and a lower model. Multiple upper models can be deployed according to the number of patches. On the other hand, one single lower model is deployed. Moreover, each upper and lower model should be placed on separate machines to avoid having all the patches in one place. If someone can access all the patches, the original image can be reconstructed. Therefore the patches are managed under strict access control. As a result of this architecture, no single machine can see a complete input image during training.

The third part is *AdaptationNet* that directly classifies features extracted from the patches. Increasing the number of upper models reduces its accuracy due to the difficulty of training multiple upper models. To mitigate this adverse effect and accelerate upper model adaptation, we place an additional network, AdaptationNet, followed by all upper models separately from a lower model, and introduce a new loss function for AdaptationNet. Since this AdaptationNet works only during training, the calculation of estimation is unaffected by this change.

We implement a prototype of our Patch SplitNN by using ResNet18 as a base model, and evaluate the performance of it with image classification tasks (CIFAR-10, CIFAR-100, and PatchCamelyon). Experiments are conducted not only for performance comparison between a base model and Patch SplitNN, but also to clarify the effect of patch size and the number of upper models. We also demonstrated the patch drop approach that sets a patch to zero randomly at each step in training process, that makes the computational load lower and patch splitting approach more secure.

Our contributions can be summarised as follows:

- We implement a prototype of Patch SplitNN which enables both privacy-preservation and lower computational cost on the client side.
- We show Patch SplitNN with AdaptationNet is very competitive with a base model (ResNet18) for some common image classification benchmarks. The AdaptationNet is shown to mitigate the adverse effect of increasing the number of upper models.
- We find the process of splitting and concatenating patches in the middle of the CNN is a highly effective data augmentation. Patch SplitNN with single upper model overcomes our base model's (ResNet18) accuracy.

2. Related Work

In this section, we briefly review related works from the areas of privacy-preserving neural network and patch representation for a vision model.

2.1. Privacy-preserving neural network

Privacy-preserving GAN [16, 23] (PP-GAN) is a variant of GAN (Generative Adversarial Network) designed for privacy preservation. The basic concept of this approach is that the GAN is designed to erase the privacy-sensitive information from an input image while maintaining useful information for a target task. The PP-GAN shows good trade-off between recognition utility and privacy protection even in real data [24], however, it is pointed out that the PP-GAN has a potential risk of the leakage of privacy-sensitive information [13]. More than that, it is sometimes tough to run a large neural network like GAN on a client device.

Federated learning [9] and split neural network (SplitNN) [22] are two popular distributed learning approaches for privacy-preserving machine learning.

Federated learning copies a global model in the central server to clients (called local model), trains the local model at each client on their local data in parallel for certain epochs and sends back gradient updates computed on each client to the server. The server creates a new global model using the updated models and shares the new global model with the clients again. This procedure continues until the model converges. The privacy-sensitive information in the input data is preserved because only model information (gradients) is shared with the central server.

SplitNN splits a deep learning network into multiple parts and these parts are trained on different devices. In the simplest setting, the base network is split into two parts, from the top to a certain layer called a cut layer, and from the cut layer to the bottom. These parts are called the client-side network and the server-side network, respectively. The clients train only the client-side network using their local data and send the output of the client-side network (an intermediate feature of the base network) to the server. The server trains only the server-side network using the intermediate features sent by the clients. The privacy-sensitive information in the input data is preserved because only intermediate features are shared with the central server.

Both federated learning and SplitNN require high computational power for the client because the clients need to run the backpropagation step which is the most computationally expensive part of the training process.

2.2. Using patches as input representation

While conventional convolutional neural network uses a raw image as an input, some recent modern neural networks, for example Vision Transformer (ViT) family [5, 14, 19] and MLP-Mixer family [12, 17, 18, 20], use patch representations as an input.

ViT family and MLP-Mixer family use patch embeddings which splits the whole input image into small regions and generate embeddings from each small region. These models achieve SOTA-level performance in the major vision tasks, like classification and object detection.

These facts imply that the patch embeddings approach is useful for a neural network for vision and cannot be a bottleneck of performance. The ConvMixer [20] also shows the potential effect of the patch embeddings and supports our approach. In this paper the patch embeddings approach is used to make training more secure and reduce the computational cost on clients.

3. Patch SplitNN

In Patch SplitNN, the main considerations are (i) how to split an image to patches (patching strategy) and (ii) how to mitigate a performance degradation due to increasing the number of upper models. Patches should be small enough to make privacy indistinguishable and must have enough information to build a model with competitive performance. Increasing the number of upper models leads to decreasing input data size or amount for each upper model, since the number of patches basically corresponds to the number of upper models for preserving privacy. This can cause difficulties when training the upper models. We present the base architecture of Patch SplitNN, then propose the patching strategy to achieve both privacy-preserving and competitive performance. In addition, for accelerating upper model training, we introduce additional layers (named **AdaptationNet**) into Patch SplitNN (named **Patch SplitNN+**) in order to reduce a negative impact of increasing the number of upper models.

3.1. Base Architecture

In Patch SplitNN, upper models are deployed in the same number as patches. If the number of upper models is less than the number of patches, the privacy level can be reduced because one upper model can take multiple patches and it can be possible to recover something closer to the original image. Figure 2 shows a case where 9 patches have been split from an original image. All outputs of upper models are concatenated before the lower model. The concatenation in Patch SplitNN is to place each patch in the same position as an original image.

Patch size is a very important parameter as much as a patch stride. If a patch size is almost the same as the original image size, privacy information must be included. On the other hand, if a patch size is too small, model performance may be much lower than expected. In addition, when a patch stride is less than a patch size, it means that a part of a patch has been overlapped with other patches. We consider a small overlap to be likely to help the model performance, because convolution can apply for boundaries among patches. To confirm our hypothesis, we propose the patching strategy to generate multiple patching patterns without any modification of a base model in Section 3.2.

Moreover, in Patch SplitNN+, all outputs of upper models can be input to AdaptationNet without any concatenation. As a result, AdaptationNet outputs 9 loss values (named *upper_loss_1* to *upper_loss_9*, respectively), then *upper_loss* is calculated based on those values. We explain the architecture of AdaptationNet and details of this new loss function in Section 3.3.

The prototype is a simple implementation to ensure the performance of Patch SplitNN and verify our hypothesis. For example, since all upper models and lower models are deployed as a single model, there is no special communication among models. The details of our implementation is presented in Section 3.4.

3.2. Patching Strategy

To minimize modification of a base model, we define 4 conditions to make patches from an original image.

- All patches are the same size.
- An aspect ratio of a patch is the same as an original image. There is no padding for each patches.
- The number of vertical and horizontal patches is the same. Therefore, the total number of patches becomes the square of a certain number.
- As long as the above conditions are met, a part of the patch may overlap with other patches, which is controlled using a stride size.

Multiple patching patterns exist even if the all above conditions are met. For example, if the parameters are an original image size ($O_{\text{size}} = (32, 32)$), a patch image size ($P_{\text{size}} = 16$) and a patch stride size ($P_{\text{stride}} = 8$), the number of patches (P_{num}) and an overlap size are *9* and *8* respectively. Our experiments show how patch patterns impact performance.

3.3. AdaptationNet

Increasing the number of upper models ($M_{\text{num}}^{\text{upper}}$) impacts model performance negatively, because the amount of data for each upper model can be decreased. To mitigate this adverse effect and accelerate upper models training, we add AdaptationNet after the upper model separately from lower

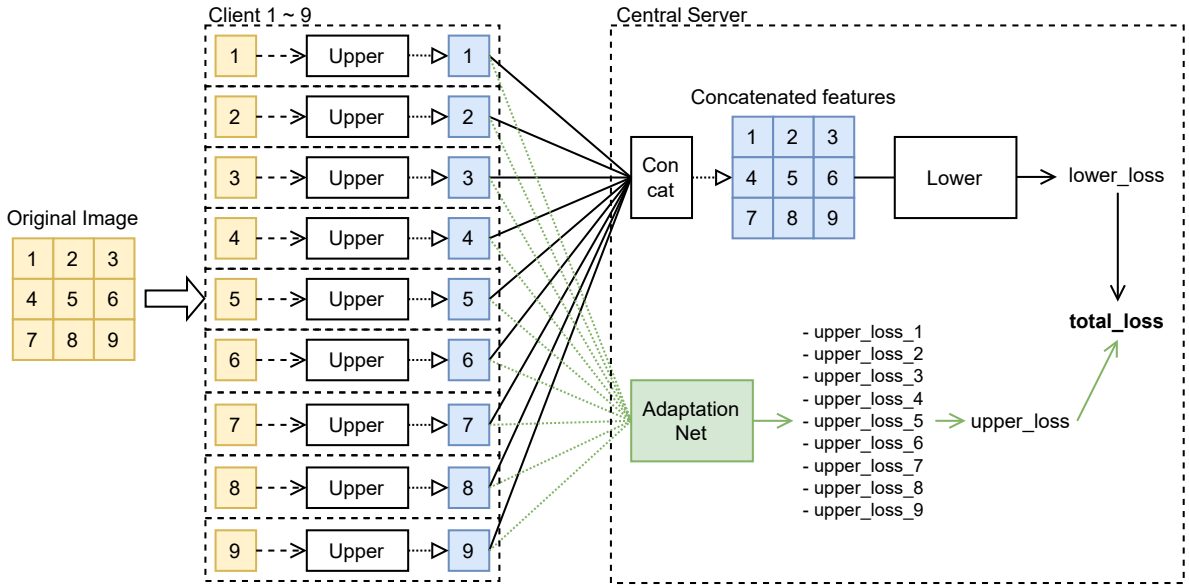

Figure 2. Details of Patch SplitNN model. Normal **Patch SplitNN** has no **AdaptationNet**. Therefore, *total_loss* is equal to *lower_loss*. In **Patch SplitNN+**, **AdaptationNet** works only training process, and *total_loss* is *lower_loss* plus *upper_loss*.

Layer	Parameters
AdaptiveAveragePooling	$ceil(output\_size/2)$
Flatten	-
Linear	$(prev\_output, 512)$
BatchNorm1d	512
ReLU	-
Linear	$(512, 256)$
BatchNorm1d	256
ReLU	-
Linear	$(256, num\_classes)$
log_softmax/sigmoid	-

Table 1. AdaptationNet is composed of the above layers. The parameter of *AdaptiveAveragePooling* is the value obtained by dividing the output size of each upper model by 2 and rounding up. The first linear layer takes the value of a *channel* size of a patch multiplied by an output size of *AdaptiveAveragePooling* ($prev\_output$) as an input parameter. Other parameters can be changed depending on the tasks. The dimension of the last layer depends on the final task.

model (see Figure 2). Table 1 shows the layer configurations of AdaptationNet. There are three linear layers after the *AdaptiveAveragePooling* layer. An output of each upper model is input to AdaptationNet without any concatenation, therefore as many outputs as inputs are calculated.

According to the introduction of AdaptationNet, we define a new loss function for *Patch SplitNN+*. The loss function consists of *total_loss* (L_{total}), *lower_loss* (L_{lower}), and *upper_loss* (L_{upper}). L_{upper} can be calculated using the following formula with the number of patches P_{num}:

$$L_{\text{upper}} = \left(\sum_{i=0}^{P_{\text{num}}} L_{\text{upper}_i} \right) \frac{M_{\text{num}}^{\text{upper}}}{P_{\text{num}}} \tag{1}$$

L_{upper_i} denotes a loss value obtained by using an output of corresponding upper model.

L_{lower} can be computed in the normal process. Then, L_{total} can be calculated with L_{lower} and L_{upper} which is multiplied by coefficient α.

$$L_{\text{total}} = L_{\text{lower}} + \alpha L_{\text{upper}} \tag{2}$$

In Patch SplitNN, there is no AdaptationNet, thereby $L_{\text{total}} = L_{\text{lower}}$.

3.4. Implementation

A prototype implementation is based on ResNet [7] architecture. Figure 3 shows the details of the upper and lower model layers. In this configuration, the upper model is from the top convolution layer to the first *Residual Blocks*, and the lower model is the rest. This setting can be changed, for example, the upper model is from the top layer to the second *Residual Block*. However, the shallow upper model may lead to weak data security for privacy-preserving and the deep upper model may lead to higher computation loads on a client machine. On the other hand, due to the advantage of the simple splitting strategy, Patch SplitNN can be implemented by using not only ResNet but also any other neural network.

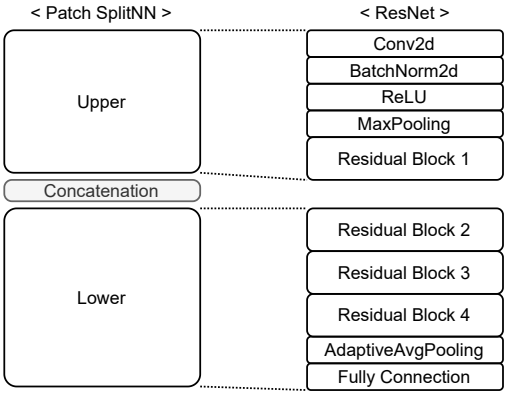

Figure 3. The layer configurations for upper and lower model based on ResNet in Patch SplitNN.

Ideally, the upper models and the lower model communicate via a network interface, since they can be placed on different machines. Since we focus on the performance check, our prototype deploys all upper and lower models into one machine as a single model. As a result of this implementation, there is no communication among models.

4. Experiments

To show that Patch SplitNN and Patch SplitNN+ have a competitive performance to a base model (ResNet18), we evaluate for image classification task with several datasets and settings. These experiments also present that Patch SplitNN+ reduces the adverse effect of deploying multiple upper models.

4.1. Datasets

We evaluate our approach on the following datasets:

CIFAR-10 [10] is a popular tiny dataset for the image classification task. This dataset is composed of 60,000 (32×32) RGB images in 10 classes. Each class has 6,000 images. Training samples are a subset of 50,000 images ($5,000 \times 10$), and test samples are 10,000 images ($1,000 \times 10$).

CIFAR-100 [10] is more difficult version of CIFAR-10. This dataset is also composed of 60,000 (32×32) RGB images in 100 classes. Each class has 600 images. Training samples are a subset of 50,000 images (500×100) and test samples are 10,000 images (100×100).

PatchCamelyon (PCam) [21] is a challenging image classification task bigger than CIFAR-10. This dataset is composed of 327,680 (96×96) RGB images with binary labels indicating the presence of metastatic tissue. Training samples are 262,144 images, and validation and test samples are 32,768 images each.

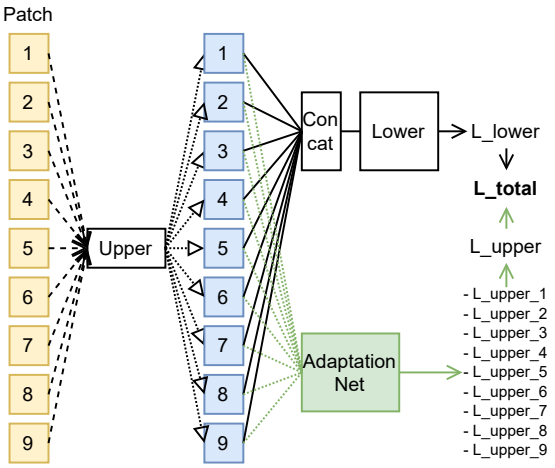

Figure 4. Overview of Patch SplitNN and Patch SplitNN+ if the number of the upper model is one. All patches are input to the single upper model sequentially, then all outputs of that upper model can be used to calculate each L_{upper_i}. L_{total} can be calculated from L_{upper_i} using Equation 1 and 2

P_{size}	P_{stride}	P_{num}	Overlap	Concat Size	Upper MACs
(16, 16)	16	4	0 (0%)	(32, 32)	38.52M
(16, 16)	8	9	8 (50%)	(48, 48)	38.52M
(14, 14)	6	16	8 (57%)	(56, 56)	29.50M
(11, 11)	7	16	4 (36%)	(44, 44)	18.21M
(12, 12)	5	25	7 (58%)	(60, 60)	21.67M
(8, 8)	8	16	0 (0%)	(32, 32)	9.63M
(8, 8)	6	25	2 (25%)	(40, 40)	9.63M
(7, 7)	5	36	2 (28%)	(42, 42)	7.37M

Table 2. Patching patterns in CIFAR-10 and CIFAR-100.

We measure performance using Top-1 classification accuracy for CIFAR-10 and CIFAR-100 and binary classification accuracy (threshold is 0.5) for PCam. Regarding PCam, we also measure AUC (Area Under Curve) as well.

4.2. Experimental Setup

Patching Patterns A patching pattern may affect the model's performance, especially depending on the level of overlap with each patch. As we describe, the number of upper models may also affect performance due to the difficulty of training. To confirm the difference among patching patterns and the impact of AdaptationNet when training with a large number of upper models, we prepare 8 patterns based on 4 conditions in the patching strategy in Table 2 and Table 3. Each patch size and patch stride of PCam are 3 times those of CIFAR because the original image size of PCam is

P_{size}	P_{stride}	P_{num}	Overlap	Concat Size	Upper MACs
(48, 48)	48	4	0 (0%)	(24, 24)	26.95M
(48, 48)	24	9	24 (50%)	(36, 36)	26.95M
(42, 42)	18	16	24 (57%)	(44, 44)	22.23M
(33, 33)	21	16	12 (36%)	(36, 36)	14.82M
(36, 36)	15	25	21 (58%)	(45, 45)	15.16M
(24, 24)	24	16	0 (0%)	(24, 24)	6.74M
(24, 24)	18	25	6 (25%)	(30, 30)	6.74M
(21, 21)	15	36	6 (28%)	(36, 36)	6.52M

Table 3. Patching patterns in PCam.

3 times larger than the size of CIFAR.

We also calculate multiply-accumulate (MACs) for each patch size. The number of MACs is quartered when patch size is halved. Therefore, Patch SplitNN can reduce computational cost on the client side.

Patch Dropping If some patches can be dropped in the training process without any degradation, it would be very helpful for further privacy preservation. Our idea is to select patches randomly and zero-fill after splitting patches. The number of patches to be dropped is the following ratio to the total number: 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, and 0.7. We evaluate the effect of dropping patches by using the best combination of conditions for each dataset.

Model and Upper Model Combinations We hypothesize that the performance of Patch SplitNN can be reduced when the number of upper models increases. AdaptationNet has the potential to mitigate the adverse effect. To verify our hypothesis and the effectiveness of AdaptationNet, our experiment includes $M_{\text{num}}^{\text{upper}} = 1$ and $M_{\text{num}}^{\text{upper}} = P_{\text{num}}$ for all patching patterns.

The model configuration for $M_{\text{num}}^{\text{upper}} = P_{\text{num}}$ is different from that for $M_{\text{num}}^{\text{upper}} = 1$. While Figure 2 presents the former one, Figure 4 shows the later one. The only difference is the number of upper models, while all else (the loss function and concatenation method) remains the same.

Parameters Table 4 shows the details of layer configurations and parameters of ResNet18 for CIFAR-10, CIFAR-100, and PCam. In CIFAR-10 and CIFAR-100, we remove *MaxPooling*, and change the first *Conv2d* parameters from their original because the image size of CIFAR is very small. On the other hand, for PCam, since the image size is much larger than CIFAR, all parameters are kept the same as a default ResNet18.

In addition, the parameters of AdaptationNet are presented in Table 1. Coefficient values are set as $\alpha = 1.0$ in CIFAR-10, $\alpha = 0.6$ in CIFAR-100, and $\alpha = 0.3$ in PCam,

Patch SplitNN	ResNet18	Params. for CIFAR	Params. for PCam
Upper	Conv2d	k=3,s=1,p=1	k=7,s=2,p=3
	BatchNorm2d	64	64
	ReLU	-	-
	MaxPooling	NA	k=3,s=2,p=1
	Layer 1	c=64	c=64
Lower	Layer 2	c=128,s=2	c=128,s=2
	Layer 3	c=256,s=2	c=256,s=2
	Layer 4	c=512,s=2	c=512,s=2
	AveragePooling2d	(1, 1)	(1, 1)
	Linear	512	512
	Last Layer	Log-softmax	Sigmoid

Table 4. Parameters of Patch SplitNN for each dataset.

which were selected in prior experiments. Other training parameters are as follows: for CIFAR-10 and CIFAR-100, $Learning\_Rate = 0.05$, $Batch\_Size = 256$, $Epoch = 200$, $Optimizer = SGD$, and $Scheduler = Onecycle$, and for PCam, $Learning\_Rate = 0.01$, $Batch\_Size = 128$, $Epoch = 60$, $Optimizer = SGD$, and $Scheduler = MultiStepLR$ (steps are $[20, 35, 50]$ and *gamma* is 0.1).

Data Augmentation The following data augmentation methods are applied for each dataset before splitting original images into patches: for CIFAR-10 and CIFAR-100, *RandomCrop* ($size = 32$ and $padding = 4$) and *RandomHorizontalFlip* ($p = 0.5$), and for PCam,*RandomCrop* ($size = 96$ and $padding = 12$), *RandomHorizontalFlip* ($p = 0.5$), *RandomVerticalFlip* ($p = 0.5$), *ColorJitter* ($b = 0.2$, $c = 0.2$, $s = 0.2$, and $h = 0.2$), and *RandomGrayscale* ($p = 0.1$).

4.3. Results

Overall The quantitative results for all datasets CIFAR-10, CIFAR-100, and PCam, and all combinations of patching patterns, models, and $M_{\text{num}}^{\text{upper}}$ are summarized in Table 5, Table 6 and Table 7. We highlight in bold any improved performance over ResNet18. These results show that the performance can be degraded from $M_{\text{num}}^{\text{upper}} = 1$ when $M_{\text{num}}^{\text{upper}}$ increases in Patch SplitNN. On the other hand, in Patch SplitNN+, that degradation can be reduced thanks to AdaptationNet. Our approach, especially Patch SplitNN+, has a highly competitive performance with a base model while reducing privacy-sensitive data.

Figures 5, 6, and 7 show that when dropping patches, model performance changes almost linearly with drop rate, where the error bars in these figures represent 1 sigma. Depending on the dataset, even with some patches dropped it still has roughly the same performance as ResNet because patches have overlaps. However, especially for PCam in $M_{\text{num}}^{\text{upper}} = 1$, dropping patch can be improve the accuracy

$(P_{\text{size}}, P_{\text{stride}})$	Model	$M_{\text{num}}^{\text{upper}}$	Top-1 Acc.
-	ResNet18	-	94.32
(16, 16)	Patch SplitNN	1	94.12
		4	93.59
	Patch SplitNN+	1	**94.38**
		4	93.56
(8, 8)	Patch SplitNN	1	93.59
		16	92.97
	Patch SplitNN+	1	94.15
		16	93.76
(16, 8)	Patch SplitNN	1	**94.74**
		9	93.68
	Patch SplitNN+	1	**94.80**
		9	**94.34**
(14, 6)	Patch SplitNN	1	**94.71**
		16	93.84
	Patch SplitNN+	1	**94.75**
		16	**94.67**
(11, 7)	Patch SplitNN	1	**94.70**
		16	93.71
	Patch SplitNN+	1	**95.20**
		16	**94.54**
(12, 5)	Patch SplitNN	1	**94.73**
		25	93.66
	Patch SplitNN+	1	**95.14**
		25	**94.81**
(8, 6)	Patch SplitNN	1	**94.50**
		25	93.42
	Patch SplitNN+	1	**95.01**
		25	**94.33**
(7, 5)	Patch SplitNN	1	**94.46**
		36	93.38
	Patch SplitNN+	1	**94.81**
		36	93.93

Table 5. CIFAR-10 Top-1 Classification Accuracy. (Average of 5 trials)

$(P_{\text{size}}, P_{\text{stride}})$	Model	$M_{\text{num}}^{\text{upper}}$	Top-1 Acc.
-	ResNet18	-	74.02
(16, 16)	Patch SplitNN	1	**74.36**
		4	73.68
	Patch SplitNN+	1	**74.22**
		4	73.83
(8, 8)	Patch SplitNN	1	72.98
		16	71.80
	Patch SplitNN+	1	73.87
		16	72.54
(16, 8)	Patch SplitNN	1	**74.72**
		9	74.00
	Patch SplitNN+	1	**75.01**
		9	**74.27**
(14, 6)	Patch SplitNN	1	**74.16**
		16	73.40
	Patch SplitNN+	1	**74.45**
		16	**74.13**
(11, 7)	Patch SplitNN	1	**74.59**
		16	73.77
	Patch SplitNN+	1	**75.52**
		16	**74.60**
(12, 5)	Patch SplitNN	1	**74.03**
		25	73.11
	Patch SplitNN+	1	**74.90**
		25	**74.06**
(8, 6)	Patch SplitNN	1	**74.41**
		25	72.84
	Patch SplitNN+	1	**75.23**
		25	**74.18**
(7, 5)	Patch SplitNN	1	**74.42**
		36	72.26
	Patch SplitNN+	1	**75.49**
		36	**74.53**

Table 6. CIFAR-100 Top-1 Classification Accuracy.(Average of 5 trials)

due to removing extra information. As a result, the dropping patch approach may be able to strengthen privacy-preserving in Patch SplitNN.

4.4. Discussion

Both Patch SplitNN and Patch SplitNN+ in $M_{\text{num}}^{\text{upper}} = 1$ and $P_{\text{size}} > P_{\text{stride}}$ are considered to be one of the powerful data augmentation methods because concatenated features have more data than original. For example, if a patching pattern is $(P_{\text{size}}, P_{\text{stride}}) = (11, 7)$ in CIFAR, the concatenated feature size is $(44, 44)$ while original output size of *layer1*, is $(32, 32)$. One evidence is that Patch SplitNN with $(P_{\text{size}}, P_{\text{stride}}) = (16, 16)$ or $(P_{\text{size}}, P_{\text{stride}}) = (8, 8)$ which has same concatenated feature size $(32, 32)$ as an original size is lower performance than ResNet18. For further improvements in privacy preservation, it is possible to deploy multiple lower models.

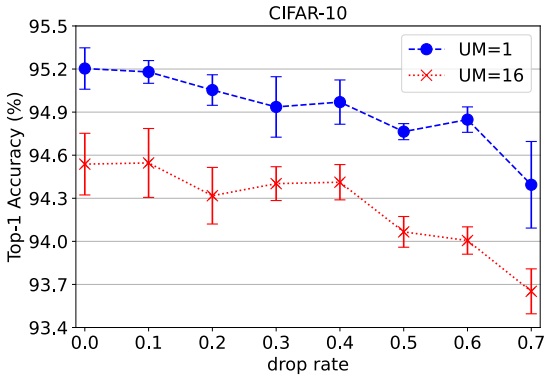

Figure 5. Patch dropping in CIFAR-10. (Average of 5 trials)

Regarding the results of patch dropping, the performance of CIFAR-100 has been more degraded than CIFAR-10.

$(P_{\text{size}}, P_{\text{stride}})$	Model	$M_{\text{num}}^{\text{upper}}$	Acc.	AUC
-	ResNet18	-	86.48	95.39
(48, 48)	Patch SplitNN	1	**86.74**	**95.69**
		4	**86.87**	**95.71**
	Patch SplitNN+	1	**86.88**	**95.65**
		4	**87.13**	**95.88**
(24, 24)	Patch SplitNN	1	86.30	95.30
		16	83.95	93.70
	Patch SplitNN+	1	85.56	95.22
		16	**86.83**	**95.44**
(48, 24)	Patch SplitNN	1	**87.09**	**95.79**
		9	**87.16**	**95.56**
	Patch SplitNN+	1	**86.63**	**95.46**
		9	**87.09**	**95.77**
(42, 18)	Patch SplitNN	1	**87.78**	**96.05**
		16	**86.73**	95.24
	Patch SplitNN+	1	**87.01**	**95.95**
		16	**87.13**	**95.86**
(33, 21)	Patch SplitNN	1	**86.93**	**95.83**
		16	**86.75**	95.24
	Patch SplitNN+	1	**87.00**	**95.84**
		16	**87.28**	**95.83**
(36, 15)	Patch SplitNN	1	**87.15**	**95.83**
		25	86.37	95.11
	Patch SplitNN+	1	**87.19**	**95.93**
		25	**87.73**	**95.95**
(24, 18)	Patch SplitNN	1	86.46	**95.48**
		25	83.90	93.67
	Patch SplitNN+	1	86.06	95.21
		25	**87.01**	**95.71**
(21, 15)	Patch SplitNN	1	**86.65**	**95.68**
		36	83.69	93.64
	Patch SplitNN+	1	**87.31**	**95.91**
		36	**87.13**	**95.66**

Table 7. PCAM Accuracy and AUC. (Average of 3 trials)

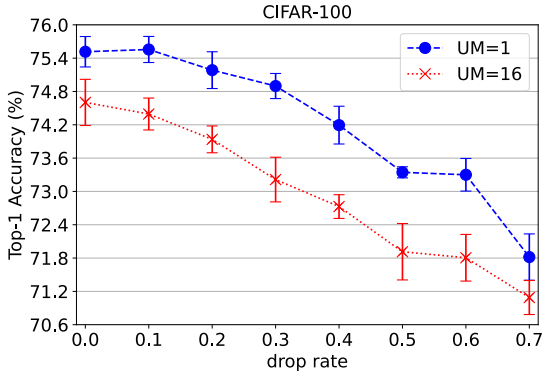

Figure 6. Patch dropping in CIFAR-100. (Average of 5 trials)

This is probably because the number of training data for each class in CIFAR-100 is smaller than CIFAR-10. Therefore, one possible improvement is to train more epochs for CIFAR-100.

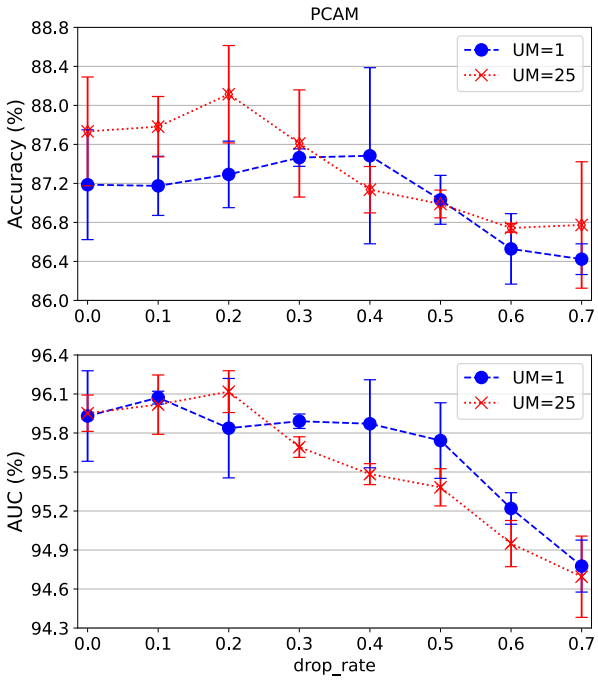

Figure 7. Patch dropping in PCam. (Average of 3 trials)

In addition, most results are straightforward, however, a part of the results in PCam is not as expected. This is probably because the number of training data in PCam is much larger than CIFAR and the standard deviation of the results is larger than CIFAR. Learning longer epochs may be necessary to avoid this.

5. Conclusion

We proposed Patch SplitNN for preserving privacy in vision tasks. Our experimental results show that Patch SplitNN and Patch SplitNN+ have competitive performance compared to ResNet18, which is used as a base model. In particular, Patch SplitNN+ outperforms both Patch SplitNN and ResNet18 in almost all patching patterns thanks to AdaptationNet, which can mitigate the negative impact of increasing the number of upper models. In addition, specifying $P_{\text{num}} = 1$ in Patch SplitNN is effective as a strong data augmentation method.

We focused on image classification in this paper, however, our approach can be extended to other tasks which use deep neural networks. Thus, adopting Patch SplitNN to other vision tasks, such as face recognition, object detection, and action recognition, is an interesting direction for future work.

References

[1] Song Bian, Tianchen Wang, Masayuki Hiromoto, Yiyu Shi, and Takashi Sato. ENSEI: efficient secure inference via frequency-domain homomorphic convolution for privacy-preserving visual recognition. *CoRR*, abs/2003.05328, 2020.

[2] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ITCS '12, page 309–325, New York, NY, USA, 2012. Association for Computing Machinery.

[3] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. Proc. Adv. in Cryptology-ASIACRYPT, 2016. https://eprint.iacr.org/2016/870.

[4] Ilaria Chillotti, Marc Joye, and Pascal Paillier. Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. Cryptology ePrint Archive, Paper 2021/091, 2021. https://eprint.iacr.org/2021/091.

[5] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ICLR*, 2021.

[6] Craig Gentry. Fully homomorphic encryption using ideal lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[8] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *CoRR*, abs/1912.04977, 2019.

[9] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtarik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. In *NIPS Workshop on Private Multi-Party Machine Learning*, 2016.

[10] Alex Krizhevsky. Learning multiple layers of features from tiny images. *MSc. Thesis*, 2009.

[11] Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. Cryptology ePrint Archive, Paper 2020/1533, 2020. https://eprint.iacr.org/2020/1533.

[12] Hanxiao Liu, Zihang Dai, David So, and Quoc V Le. Pay attention to mlps. In *Advances in Neural Information Processing Systems*, volume 34, pages 9204–9215, 2021.

[13] Kang Liu, Benjamin Tan, and Siddharth Garg. Subverting privacy-preserving gans: Hiding secrets in sanitized images. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 14849–14856. AAAI Press, 2021.

[14] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2021.

[15] Maarten G. Poirot, Praneeth Vepakomma, Ken Chang, Jayashree Kalpathy-Cramer, Rajiv Gupta, and Ramesh Raskar. Split learning for collaborative deep learning in healthcare. *CoRR*, abs/1912.12115, 2019.

[16] Zhongzheng Ren, Yong Jae Lee, and Michael S. Ryoo. Learning to anonymize faces for privacy preserving action detection. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I*, volume 11205 of *Lecture Notes in Computer Science*, pages 639–655. Springer, 2018.

[17] Ilya Tolstikhin, Neil Houlsby, Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers, Jakob Uszkoreit, Mario Lucic, and Alexey Dosovitskiy. Mlp-mixer: An all-mlp architecture for vision, 2021.

[18] Hugo Touvron, Piotr Bojanowski, Mathilde Caron, Matthieu Cord, Alaaeldin El-Nouby, Edouard Grave, Armand Joulin, Gabriel Synnaeve, Jakob Verbeek, and Hervé Jégou. Resmlp: Feedforward networks for image classification with data-efficient training, 2021.

[19] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *CoRR*, abs/2012.12877, 2020.

[20] Asher Trockman and J. Zico Kolter. Patches are all you need? *CoRR*, abs/2201.09792, 2022.

[21] Bastiaan S Veeling, Jasper Linmans, Jim Winkens, Taco Cohen, and Max Welling. Rotation equivariant CNNs for digital pathology. June 2018.

[22] Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep

learning without sharing raw patient data. In *ICLR AI for social good workshop*, 2019.

[23] Zhenyu Wu, Zhangyang Wang, Zhaowen Wang, and Hailin Jin. Towards privacy-preserving visual recognition via adversarial training: A pilot study. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XVI*, volume 11220 of *Lecture Notes in Computer Science*, pages 627–645. Springer, 2018.

[24] Zuobin Xiong, Wei Li, Qilong Han, and Zhipeng Cai. Privacy-preserving auto-driving: a gan-based approach to protect vehicular camera data. In *2019 IEEE International Conference on Data Mining (ICDM)*, pages 668–677. IEEE, 2019.

[25] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), jan 2019.