École Polytechnique Fédérale de Lausanne

Image Obfuscation for Privacy-Preserving Machine Learning

Semester project

Author:
Louis LANDELLE

Supervisors:
Dr. Radhakrishna Achanta
Dr. Mathias Humbert

March 30, 2025

Contents

1 Abstract

When providing sensitive data to perform machine learning on untrusted platforms, a solution to preserve privacy is to obfuscate the data, with techniques chosen in such a way that chosen ML algorithms are still capable of training directly on the obfuscated dataset. *Mixup* was introduced as a way of regularizing deep networks doing image classification [1]. We note that *Mixup* could contribute towards privacy protection of the data used for training neural networks, via obfuscating samples by mixing them together before training. The original *Mixup* picks weights from a random distribution, which cannot guarantee visual obfuscation of every mixture. We instead explore a variation, *Fixed-mixup*, which does not pick weights from a random distribution. We present results for classical datasets and models when maximizing obfuscation, and also when maximizing test accuracy, for which this method can be used, and generalize the method to mix more than two images to increase privacy, and discuss results. We also explore another image visual obfuscation technique, *Pixel-shuffling*, where we shuffle sample pixels, and explore its interaction with *Fixed-mixup* to improve its fitness as an obfuscation mecanism. We finally discuss a last viable obfuscation technique, *Pixel-grafting*, inspired by *Fixed-mixup*, and discuss the results.

2 Introduction

Mixup was presented as a technique to increase the performance of neural network training, by mixing two samples together. On the other hand, as it is a common scenario these days to use cloud computing services for example, to train deep learning models for tasks like image recognition, the data used for training can not be sent over to untrusted servers in order to prevent breach of privacy. In such a case, there are several privacy protecting methods of training that may be useful. In most cases, methods of differential privacy [2], among others, come with a privacy versus utility trade-off - the better the privacy protecting nature of the training scheme, the poorer is the utility or accuracy of the model. One goal of this project is to explore the fixed variation of *Mixup* which is analogous to *mixtures* techniques for data protection. We want to confirm the effect of *mixup*, generalize it for mixing N samples instead of 2, and give a general overview of how fixing *Mixup* affects performance in multiple scenarios. We also look into other image obfuscation ideas such as *Pixel shuffling* and *Pixel grafting*, to present a general empirical overview on how various image obfuscation techniques affect the performance of model training.

In Figure 1 you can observe a generic protocol that could be used to perform such training, this project will explore what advantages and drawbacks a handful of methods could provide at the *obfuscate* step.

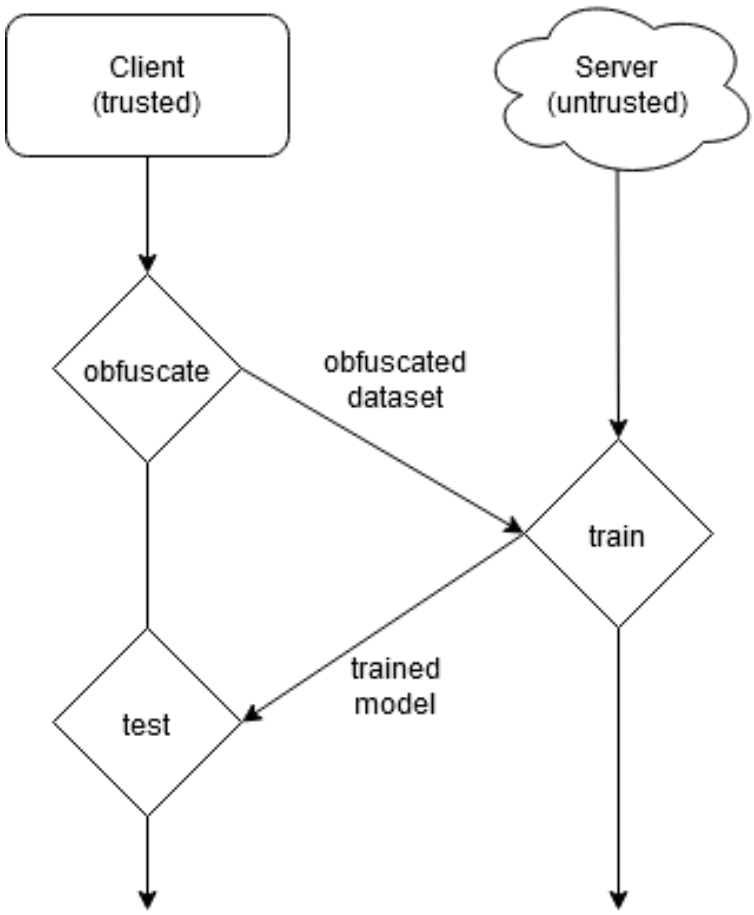


Figure 1: Generic protocol where training on obfuscated data is useful

Mixup is a simple technique, which in an unintuitive manner, improves classification accuracy. We note that MixUp "pollutes" the original image and can be considered to be a means of performing differential privacy. The client can provide mixed-up images, the mixing ratio, and the labels of the mixed-up classes to the server. The server trains a model and returns this model to the client, who can then use this model to classify "unmixed" i.e pure images.

However, it is noteworthy that unlike conventional privacy protection schemes that reduce utility, MixUp tends to improve utility [1]. Therefore, to fully explore the potential of MixUp for privacy protection, in this report, we perform a wide range of experiments.

2.1 Training setups

2.1.1 CNN with Fashion-MNIST

The first model that will be used is a small CNN, to generate results quickly, inspired by LeNet [3]. We alter it by using a single fully-connected layer and using batch norm layers.

$$INPUT \rightarrow CONV \rightarrow BATCHNORM \rightarrow RELU \rightarrow POOL \rightarrow$$

$$CONV \rightarrow BATCHNORM \rightarrow RELU \rightarrow POOL \rightarrow FC$$

We will apply this model on Fashion-MNIST, which provides us with 60000 samples for training and 10000 samples for testing [4].

2.1.2 ResNet18 with CIFAR10

The second model that will be used is the ResNet18 architecture. This model has the benefit to be able to be deeper than classical CNNs, deeper models are usually harder to train but using the technique of adding the output of the previous layer to the next (the *residual*), the model can be extended by many layers without getting too hard to train. [5] We use the 18-layers deep version. on CIFAR10 and CIFAR100. [6].

2.1.3 ResNet18 with CIFAR100

We also use the ResNet18 model on the CIFAR100 dataset [6]. With 10 times more classes, using the same number of training epochs as for FMNIST and CIFAR10 will yield poor accuracy - however, we are interested in how the techniques presented affect test accuracy, regardless of benchmark performance.

2.2 Performance evaluation

The results in the subsequent parts will show how using various techniques alter test accuracy on the models and datasets described above. In order to describe the alterations of the techniques, we need a clear benchmark on a fixed setup. The setup chosen for our experiments will be the following:

- Batch size:
  - 500 for CIFAR10 and CIFAR100
  - 600 for Fashion-MNIST
- Epochs: 100
- Learning rate: 0.001

The performance metric that we will use throughout the project is the test accuracy for the classification task:

$$Acc = 100 \times \frac{\sum TP + \sum TN}{\sum Total\ population}$$

3 Benchmark

We trained the models on the datasets 3 times to remove some variance by averaging the accuracies, and got the following results:

Item	Test accuracies			Mean test accuracy
FMNIST+CNN	90.11	89.56	89.78	89.82
CIFAR10+ResNet18	84.74	85.38	85.38	85.17
CIFAR100+ResNet18	53.86	55.69	54.93	54.83

Table 1: Benchmark results for our chosen models and datasets

4 Exploring MixUp

4.1 MixUp - what it is, how is works

Mixup is described as a method that transforms training samples with:

$$\begin{aligned} \tilde{x} &= \lambda x_i + (1-\lambda)x_j \\ \tilde{y} &= \lambda y_i + (1-\lambda)y_j \end{aligned} \tag{1}$$

The λ are generated by $\lambda \sim Beta(\alpha, \alpha)$.

Instead of trying to minimize the empirical risk defined as:

$$R_\delta(f) = \frac{1}{n}\sum_{i=1}^{n} l(f(x_i), y_i)$$

mixup trains by minimizing the Vicinal empirical risk (to perform Vicinal Risk Minimization [7])

$$R_v(f) = \frac{1}{m}\sum_{i=1}^{m} l(f(\tilde{x_i}), \tilde{y_i})$$

The paper provided data showing how this simple to implement method increases the performance of training on samples of classical models and datasets. Our first question was: can you fix λ for training and get similar performance? This would open the door for choosing λ for our own obfuscation liking.

4.2 *Mixup* and obfuscation

The system described in Equation 1 describes a process akin to a Linear instantaneous (LI) mixture. Obfuscation of images within a mixed-up result would offer privacy as long as recovering the original images is hard. The process of reversing such mixtures is the **Blind Source Separation (BSS)** problem.

Algorithms for BSS are designed to separate the sources from a system of mixtures, the "cocktail-party problem", without prior knowledge of any source.

Typically, the problem setups mixtures $\tilde{x_0}...\tilde{x_N}$ as row vectors of the matrix M known to have been generated by the process $AX = M$, where A is a $N \times N$ full rank random matrix, the "mixing matrix". The methods of BSS provide estimations to recover X, the matrix where row vectors are the source signals, such as ICA [8].

In the *mixup* case, as Equation 1 constructs the virtual sample (the mixtures) from randomly selected samples over the whole dataset, by mixing N images of an L sized dataset, we have in our resulting dataset N mixtures for each original sample. Therefore the information to perform BSS is present within the dataset. However, in a scenario where mixtures leak, the attacker has no knowledge of **which** mixtures are generated by which samples. This forces the attacker to perform BSS with A a $L \times L$ mixtures matrix, which will be sparse, as only N mixture weights are non-zero in order to separate the sources, with L the length of the entire dataset.

This is good because even with the attacker knowing the mixing weights ($\hat{\lambda}$ as will be discussed in Subsection 4.5), the mixing matrix A remains unknown as which components are 0 and which are some λ_i is not revealed.

An alternative would be, knowing N and $\hat{\lambda}$, to run FastICA with every combination of N mixtures, with a mixing matrix sized $N \times N$, and reviewing the results until we fall on the right N mixtures all containing the same source that would be properly separated. However, this method produces $\frac{L!}{(L-N)!}$ combinations to search for N viable sets of N mixtures that can be separated. As $N \uparrow$, the defender can force a very costly computation process, i.e the defender is provided with the tool to increase N (and/or L) to increase the privacy of the sources.

The implementation of images *mixup* uses the following core logic:

```
batch_size = x.size()[0]
index = torch.randperm(batch_size)
mixed_x = lam * x + (1 - lam) * x[index, :]
y_a, y_b = y, y[index]
```

The loss function to perform the Vicinal Risk Minimization which should power the *mixup*-based performance improvements is implemented in the following way:

```
pred = model(mixed_x)
loss = lam * criterion(pred, y_a) + \
    (1 - lam) * criterion(pred, y_b)
loss.backward()
```

The criterion to compose the loss is the cross-entropy loss criterion, imple-

mented in pytorch as torch.nn.CrossEntropyLoss.

4.3 *Fixed-mixup*

As discussed, the original paper's approach was to use lambdas picked from a Beta distribution of parameters α, $\beta(\alpha, \alpha)$ to produce the Vicinal Risk Minimization (VRM) principle [7]. One side effect of this approach is that as the original mixup viscinal distribution is computed, as $n \to \infty$, the usage of the symmetric Beta distribution $\beta(\alpha, \alpha)$ will average results towards the mean lambda of $\bar{\lambda} \to \frac{\alpha}{\alpha+\alpha} = 0.5$. But the other effect which is problematic, is that as λ is picked from a random distribution, it becomes possible to store mixtures that are not or only slightly visualy obfuscated. We adapt *mixup* to remove this scenario, and keep mixing ratios at our desired level.

Instead of exploring the mixup principle by picking lambda from a random distribution, we experiment on fixed lambda values, for a technique that will be referred as *Fixed-mixup* and of hyperparameter λ, and report the following results.

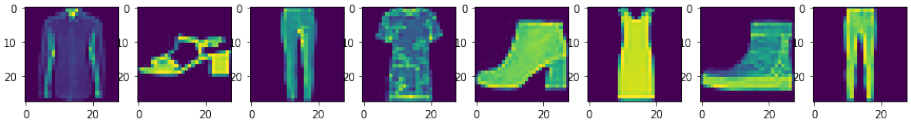
4.4 Initial results

We initially run *Fixed-mixup* in a grid-search manner with values

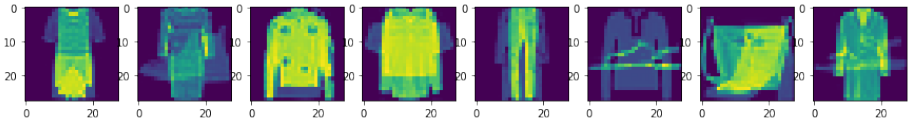
$$\lambda \in \{1.0, 0.875, 0.75, 0.625, 0.5\}$$

to understand the relationship between a fixed λ and the resulting test accuracy. For *Fixed-mixup* with two samples, the weights are λ and $1 - \lambda$. By definition, *Fixed-mixup* has a mirror symmetry around axis $\lambda = 0.5$, i.e we will get the same results for $\lambda = 0.875$ and $\lambda = 0.125$, therefore we can restrict our grid-search to $[0.5, 1.0]$ instead of searching the entire range $[0.0, 1.0]$.

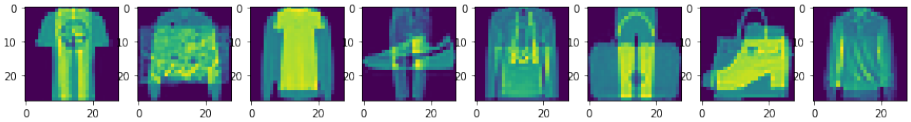
To put those results into perspective, you can observe in Figure 2 and Figure 3 what the samples look like after the application of *Fixed-mixup*:



(a) FMNIST original samples

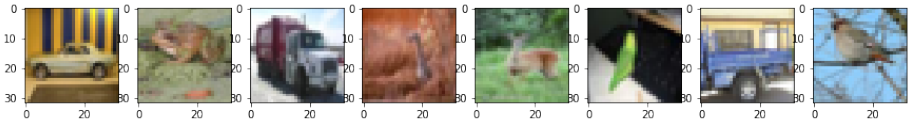


(b) FMNIST samples under *Fixed-mixup* with $\lambda = 0.75$

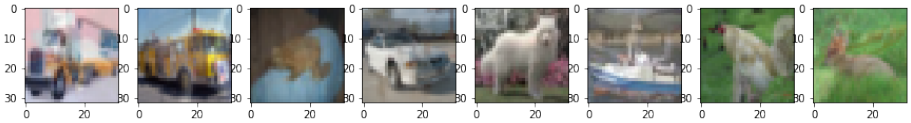


(c) FMNIST samples under *Fixed-mixup* with $\lambda = 0.5$

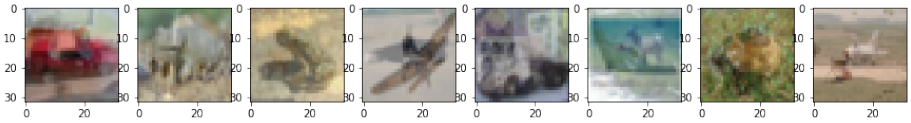
Figure 2: *Fixed-mixup* effect on FMNIST samples for different weights



(a) CIFAR10 original samples



(b) CIFAR10 samples under *Fixed-mixup* with $\lambda = 0.75$



(c) CIFAR10 samples under *Fixed-mixup* with $\lambda = 0.5$

Figure 3: *Fixed-mixup* effect on CIFAR10 samples for different weights

We report the following results:

	0	1	2	mean
$[1., 0.]$	89.97	88.96	89.96	89.63
$[0.875, 0.125]$	91.58	91.33	91.14	91.35
$[0.75, 0.25]$	91.59	91.56	91.35	91.5
$[0.625, 0.375]$	91.12	91.06	90.69	90.9567
$[0.5, 0.5]$	91.00	90.66	90.33	90.6633

Table 2: Results for FMNIST+CNN on λ grid-search

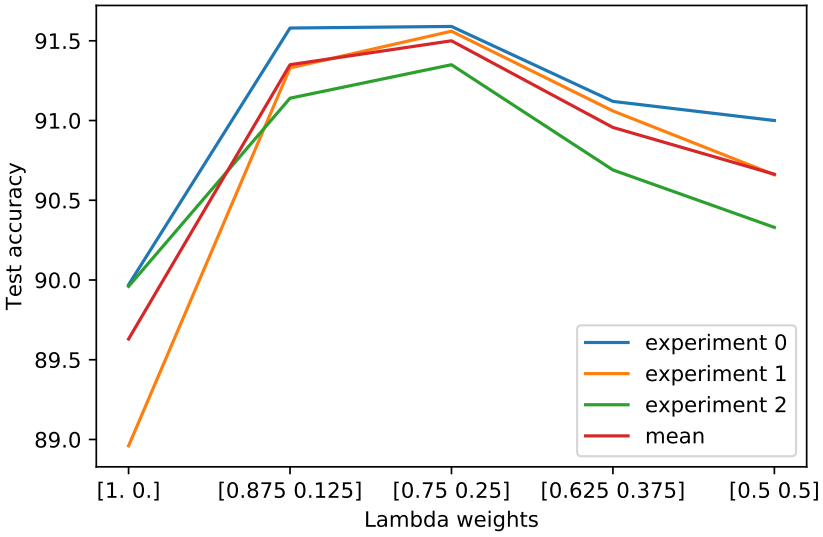


Figure 4: Test accuracy of FMNIST+CNN in relation with λ

	1	2	mean
$[1., 0.]$	84.74	85.38	85.060
$[0.875, 0.125]$	87.85	87.10	87.475
$[0.75, 0.25]$	88.55	88.25	88.400
$[0.625, 0.375]$	85.27	85.96	85.615
$[0.5, 0.5]$	82.37	83.76	83.065

Table 3: Results for CIFAR10+ResNet18 on λ grid-search

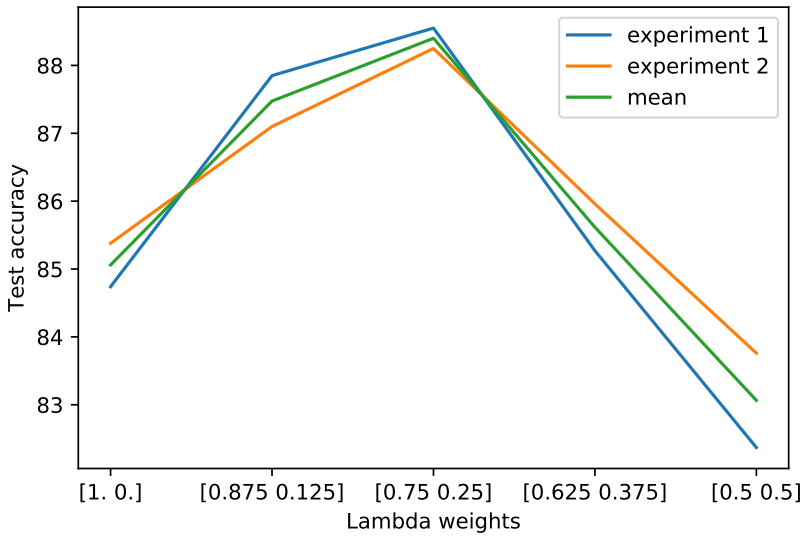


Figure 5: Test accuracy of CIFAR10+ResNet18 in relation with λ

We observe that *Fixed-mixup*, like its counterpart *mixup*, increases mean test accuracy. All lambdas are not equal however, we observe the existence of a peak in the mean test accuracy curve at the $[0.75, 0.25]$ point. We will refer to this value in the rest of this report as *"sweet spot"*, $\lambda_{sweetspot}$.

We also see different results for the $[0.5, 0.5]$ point. For FMNIST+CNN, the test accuracy here is still greater than without using mixup. For CIFAR10+ResNet18 however, this point actually presents worse test accuracy compared to no mixup. We will refer to this value in the rest of this report as *"maximum obfuscation"*, λ_{maxobf}.

This is good news because it means that we can freely choose a fixed mixup ratio to choose how much to benefit from the test accuracy increase of *mixup*, or how much to obfuscate.

We conclude that $\lambda = 0.5$ is a bad target for lambdas if the goal is to maximize performance, where fixed λ values chosen around "sweet spot" regions in turn consistently increased test accuracy. In this technique, it is λ itself that is a hyperparameter.

As regular *mixup* showcases $\bar{\lambda} \to 0.5$, we will note that for the effort of maximizing test accuracy, it could be interesting to perform further research on lambdas picked from asymmetric distributions with $\bar{\lambda} \to 0.75$. In our case we

continue exploring *Fixed-mixup* but instead of mixing two samples, we generalize mixing for N samples.

4.5 Generalized *Fixed-mixup*

As we seek to generalize *Fixed-mixup* to provide the defender with the tool to increase N to increase privacy of the sources, we call mixing weights for each sample λ_i, components of the *lambda-vector* $\vec{\lambda}$. We observe the following constraints on λ for N=2:

$$\lambda_i \in [0; 1], i \in \{1, 2\}$$

$$\lambda_1 + \lambda_2 = 1$$

For an arbitrary N we generalize the constraints to:

$$\lambda_i \in [0; 1], i \in \{1, N\} \tag{2}$$

$$\sum_{i=1}^{N} \lambda_i = 1 \tag{3}$$

We adapt the implementation of images mixing to the following:

```
perms = get_permutations(N, X.shape[0])
X_permutations = torch.stack([X[p] for p in perms], 0)
X_mixed = (X_permutations * lambda_).sum(axis=0)
ys = [y[perm] for perm in perms]
```

And the loss function to perform the Vicinal Risk Minimization in the following way:

```
preds = model(X_mixed)
loss = sum(lambda_v[i] * criterion(preds, ys[i]) \
    for i in range(lambda_v.shape[0]))
loss.backward()
```

We keep the cross-entropy loss criterion.

Now, instead of a λ hyperparameter we need a $\vec{\lambda}$ hyperparameter. This hyperparameter can be searched within a set that is increasingly large as $N \uparrow$. We observe that ordering of weights is irrelevant and only their value is important, with $\vec{\lambda}_a = [..., \lambda_1, ..., \lambda_2, ...]$ and $\vec{\lambda}_b = [..., \lambda_2, ..., \lambda_1, ...]$ having equal performance impact. Thus we can imagine $\vec{\lambda}$ as always **sorted by decreasing weights**.

Therefore, the degree of freedom we have in "how do we search the space of possible $\vec{\lambda}$" can be translated as "which decreasing function respecting our constraints should we use for $\vec{\lambda}$".

For the rest, we use the convention:

$$\hat{\lambda}_{\mathbf{nomix}} : \lambda_i = 1\{i = 0\} \tag{4}$$

This lambda vector is equivalent to training on raw samples without any *mixup*.

$$\hat{\lambda}_{\mathbf{maxobf}} : \lambda_i = \frac{1}{N} \forall i \tag{5}$$

This lambda vector mixes N samples with equal weight.

We also use $\hat{\lambda}$ instead of $\vec{\lambda}$ to when $\hat{\lambda}$ respects the constraints (2) and (3).

We can note factories for $\hat{\lambda}$s using such fonctions $f : f(N, L) = \{\hat{\lambda}_i\}_{i=0}^{L}$ where N is the number of components in each $\hat{\lambda}$ and $L + 1$ is the norm of the output set. We design the factories to produce sets of lambda vectors searching the space between $\hat{\lambda}_{\mathbf{nomix}}$ and $\hat{\lambda}_{\mathbf{maxobf}}$ and shaped according to our function chosen. Note that the sets contain the $\hat{\lambda}_{\mathbf{maxobf}}$, so L can be seen as the real amount of $\hat{\lambda}$ lying in $(\hat{\lambda}_{\mathbf{nomix}}, \hat{\lambda}_{\mathbf{maxobf}})$.

In this paper we will take a look at three of such decreasing functions, parametrized by a variation factor which allow us to produce various $\vec{\lambda}$ from the same shape function. From our decreasing functions we then generate L+1 $\vec{\lambda}$ to grid-search the optimal hyperparameter. We provide a name and the code to generate L+1 $\vec{\lambda}$ from such functions below. Note that the L+1 lambdas are designed to produce the "constant" lambda vector $\hat{\lambda} : \lambda_i = \frac{1}{N}$, view L as the number of interpolated lambdas between.

- Linearly decreasing: for L lambdas needed, we use the set of starting and ending points:

  $$P = \{(1 - \frac{k}{2L}, \frac{k}{2L})\}_{k=0}^{N}$$

  and we define our $\vec{\lambda}$ by interpolating N points evenly spaced on the line (p_0, p_1) for each $(p_0, p_1) \in P$. In this configuration, k serves as the steepness factor, where $k = 0$ will produce the $y = -x$ line and $k = L$ will produce the $y = \frac{1}{2}$ line.

  ```
  LAMBDAS = [np.interp(
      np.linspace(0., 1., N), [0., 1.], [lam, 1-lam])\
      for lam in np.linspace(1., .5, L+1)][:-1]
  ```

- Inverse power law: defined as $\vec{\lambda} = [\frac{1}{x^0}, ..., \frac{1}{x^{(N-1)}}]$, with x the steepness factor ($x = 5$ will produce the steepest $\vec{\lambda}$ and $x = 1$ will produce $y = x$-type $\vec{\lambda}$).

```
LAMBDAS = [np.array([1/(x**n) for n in range(N)])\
    for x in np.linspace(1., 6., L+1)]
LAMBDAS.reverse()
```

- Sigmoid decreasing: the weights are the points lying on the sigmoid curve if $\{0, ...N\}$ is mapped to $[-2y, +2y]$, for y in $(0, L+1)$. As y $\rightarrow$ L+1, the first weights get larger and the last weights get smaller.

```
sigmoid = lambda x: 1/(1+np.exp(-x))
lspace = lambda x, N: np.linspace(-x, x, N)
LAMBDAS = [np.array([sigmoid(x) for x in lspace(y*2, N)])[::-1]\
    for y in range(L+1)]
```

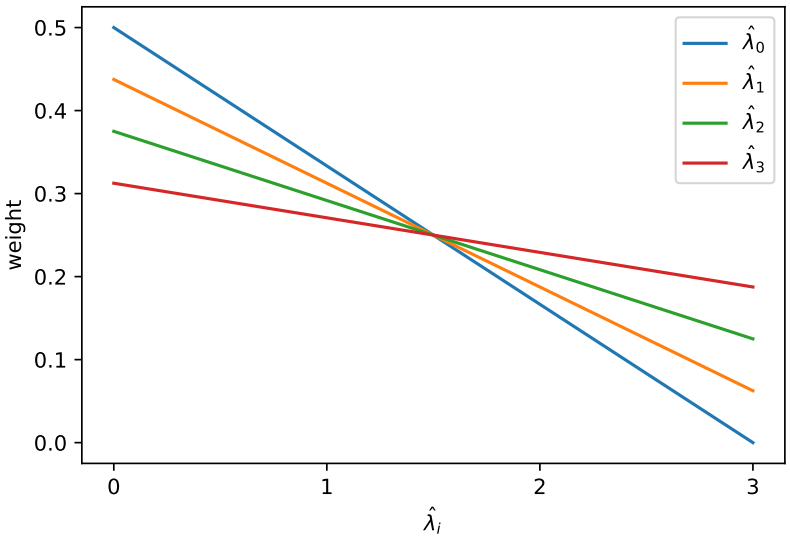


Figure 6: Linear lambda vector weights

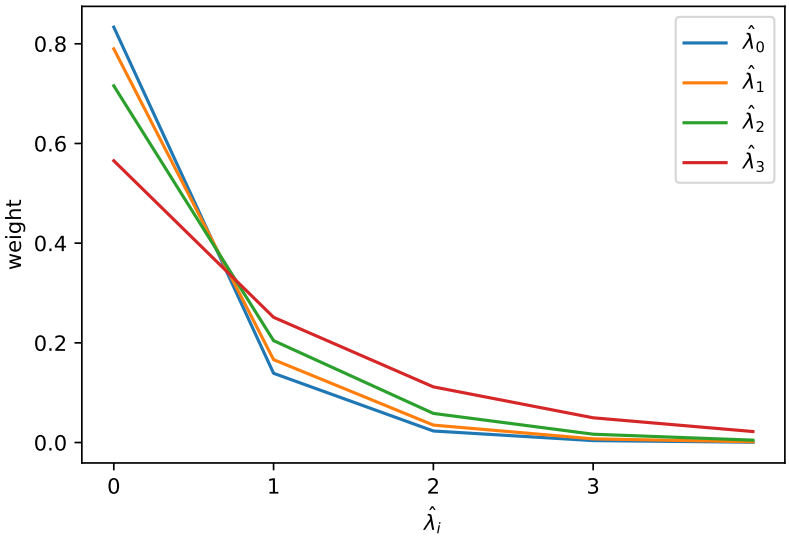


Figure 7: Inverse power law lambda vector weights

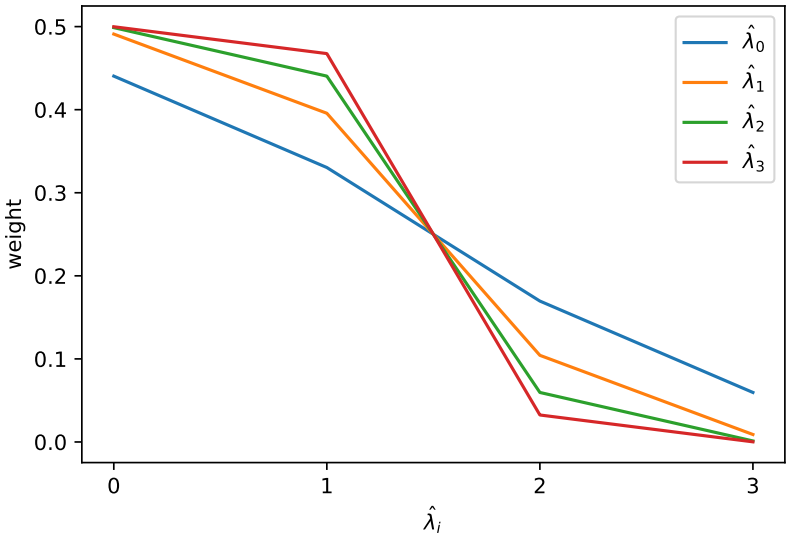


Figure 8: Sigmoid lambda vector weights

Note that such $\vec{\lambda}$ do not respect our constraint of $\sum_{i=1}^{N} \lambda_i = 1$. We can

always finish a factory of $\vec{\lambda}$ with:

$$\hat{\lambda} = \frac{\vec{\lambda}}{||\vec{\lambda}||_1}$$

We emphasize that this choice is an **arbitrary** attempt to explore the space of possible $\vec{\lambda}$.

We present the results in the tables below:

	FMNIST+CNN	CIFAR10+ResNet18	CIFAR100+ResNet18
$[0.5, 0.33330.16670.]$	90.09	84.9900	51.96
$[0.4375, 0.3125, 0.1875, 0.0625]$	90.04	83.1150	50.90
$[0.375, 0.2916, 0.2083, 0.125]$	89.04	81.4950	46.90
$[0.3125, 0.2708, 0.2292, 0.1875]$	88.79	78.2475	44.40

Table 4: Mean test accuracy for Linear $\hat{\lambda}$ factory

	FMNIST+CNN	CIFAR10+ResNet18	CIFAR100+ResNet18
$[0.8340, 0.1390, 0.0232, 0.0039]$	91.340000	87.206	57.605
$[0.7910, 0.1665, 0.0351, 0.0074]$	90.916667	87.782	58.130
$[0.7191, 0.2055, 0.0587, 0.0168]$	91.496667	87.800	57.390
$[0.5781, 0.2569, 0.1142, 0.0508]$	90.566667	86.892	55.560

Table 5: Mean test accuracy for Inverse Power Law $\hat{\lambda}$ factory

	FMNIST+CNN	CIFAR10+ResNet18	CIFAR100+ResNet18
$[0.4404, 0.3304, 0.1696, 0.0596]$	89.936	84.0375	50.01
$[0.4910, 0.3957, 0.1043, 0.0090]$	90.566	83.6975	52.44
$[0.4988, 0.4404, 0.0596, 0.0012]$	90.628	83.1050	52.33
$[0.4998, 0.4675, 0.0325, 0.0002]$	90.604	82.6200	51.19

Table 6: Mean test accuracy for Sigmoid $\hat{\lambda}$ factory

The mean test accuracies present in those data are not all computed the same, due to a variable amount of experiments ran for each combination of dataset, model and factory. Those numbers with two decimals are single experiments.

Here is how many experiments were averaged for those test accuracies, for resp. Linear, Inverse power and Sigmoid factories:

- **FMNIST+CNN**: 1, 3, 5

- **CIFAR10+ResNet18**: 4, 5, 4
- **CIFAR100+ResNet18**: 1, 2, 1

CIFAR10+ResNet18 was the main focus to generate as much data as possible, and the results are presented in Figure 9.

We have placed the line corresponding to average *Fixed-mixup* on the figure for comparaison. To interpret the three-part graph, as we go from $\hat{\lambda_0}$ to $\hat{\lambda_4}$ for each factory, we increase visual obfuscation by getting closed to $\hat{\lambda}_{maxobf}$. From those results we draw the following conclusions:

We determine that the best $\hat{\lambda}$ shape function is the most performant **Inverse power law** function, with every lambda from the factory outperforming nomix. Linear lambda provides moderate amount of obfuscation but only the first lambda vector performs similarly to nomix, with performance dropping quickly. The sigmoid factory had a generally slower decreasing performance with increasing obfuscation compared to the linear factory, this might be because as we increase obfuscation, the sigmoid shape will make the first two weights approx. 0.5 and the last two weights approx. 0, which is maximum obfuscation but for $N = 2$. Indeed, results for the last vector are similar to those found in Figure 5.

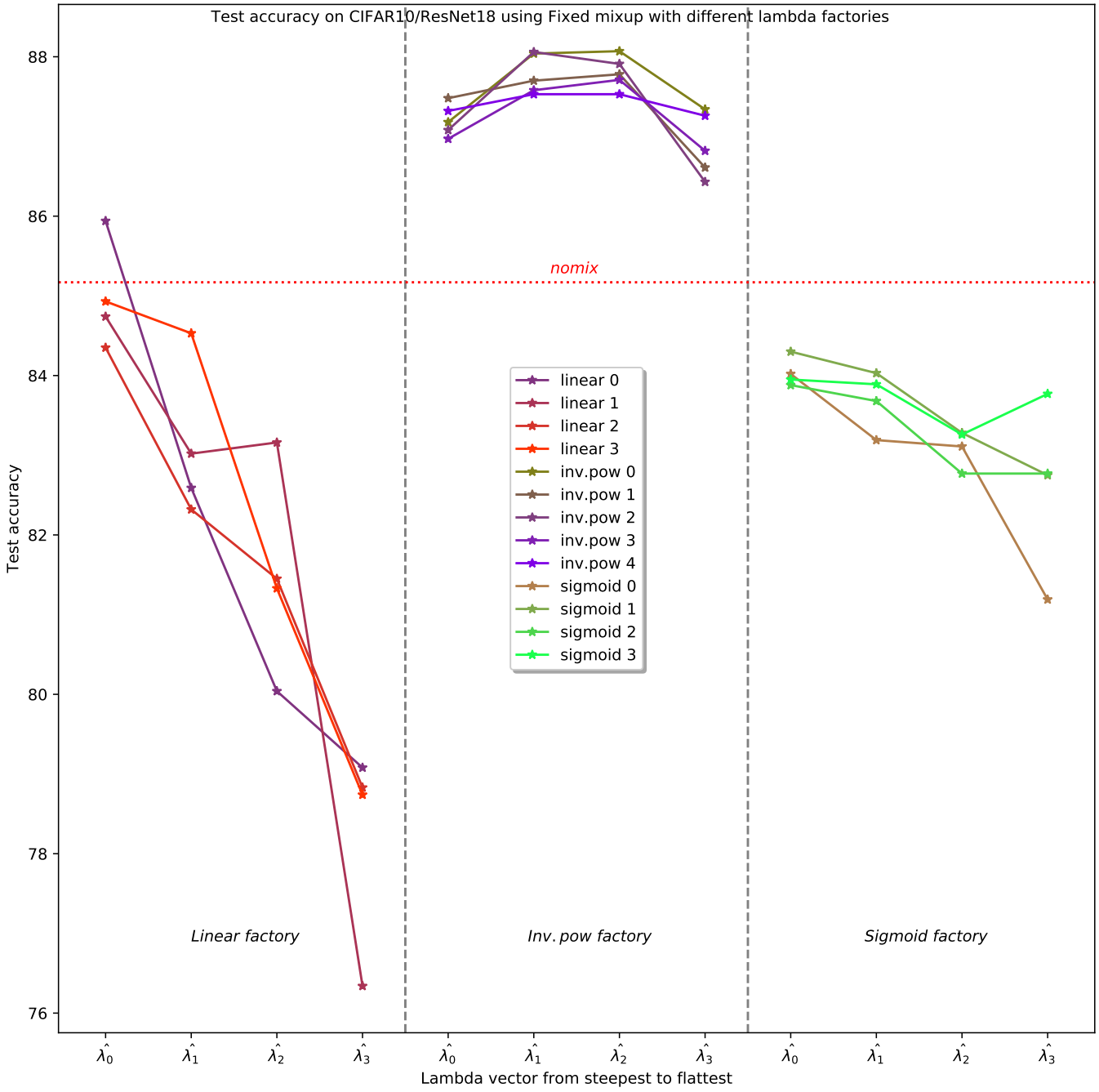


Figure 9: Test accuracy for our three factories

4.6 Multivariate *sweetspot*

The existence of a *sweetspot* for fixed $\hat{\lambda}$ we found for $N = 2$ could be generalized for $N > 2$. With the test accuracy results from above, we observe for our various setup that inverse power law $\hat{\lambda}$ always outperform our other factories for $\hat{\lambda}$. Besides, the shape of the trend of mean test accuracy for each factory is different. We observe that:

- The trend for linear $\hat{\lambda}$ is sharply decreasing with $\hat{\lambda}$ getting towards $\hat{\lambda}_{maxobf}$
- The trend for inverse power $\hat{\lambda}$ is peaking between the two intermediate values of $\hat{\lambda}$
- The trend for sigmoid $\hat{\lambda}$ is slightly decreasing with $\hat{\lambda}$ getting towards $\hat{\lambda}_{maxobf}$

We can exploit this fact to make a new factory for $\hat{\lambda}$ that would maximize test accuracy, using the "sweetspot" characteristic of *Fixed-mixup*. We notice that the interpolation of both intermediate $\hat{\lambda}$ has a first weight around 0.75, exactly similar to the $N = 2$ case.

A postulate following this similarity would be that 0.75 is a good first weight. Extending the "sweetspot" of $N = 2$ of:

$$\hat{\lambda}_{sweetspot} = [\frac{3}{4}, \frac{1}{4}]$$

We also notice that for $N = 4$, the interpolation of both intermediate $\hat{\lambda}$ is located around:

$$\hat{\lambda}_{sweetspot} = [\frac{3}{4}, \frac{3}{4^2}, \frac{3}{4^3}, \frac{1}{4^3}] = [0.75, 0.1875, 0.046875, 0.015625]$$

Which corresponds to taking the initial weights of $[\frac{3}{4}, \frac{1}{4}]$ and extending $N = 4$ times by replacing the last weight $\hat{\lambda}_i$ by $[\frac{3\hat{\lambda}_i}{4}, \frac{\hat{\lambda}_i}{4}]$ - an advantage of this construction is that it preserves naturally constraints (2) and (3).

We propose the following definition for a N-dimensional $\hat{\lambda}_{sweetspot}$ that increases test accuracy over $\hat{\lambda}_{nomix}$ using *Fixed-mixup*.

$$\hat{\lambda}_{sweetspot} : \lambda_i = \begin{cases} \frac{3}{4^{i+1}}, & \text{if } i < N-1 \\ \frac{1}{4^{N-1}}, & \text{otherwise} \end{cases} \tag{6}$$

We illustrate what the weights look like in Figure 10

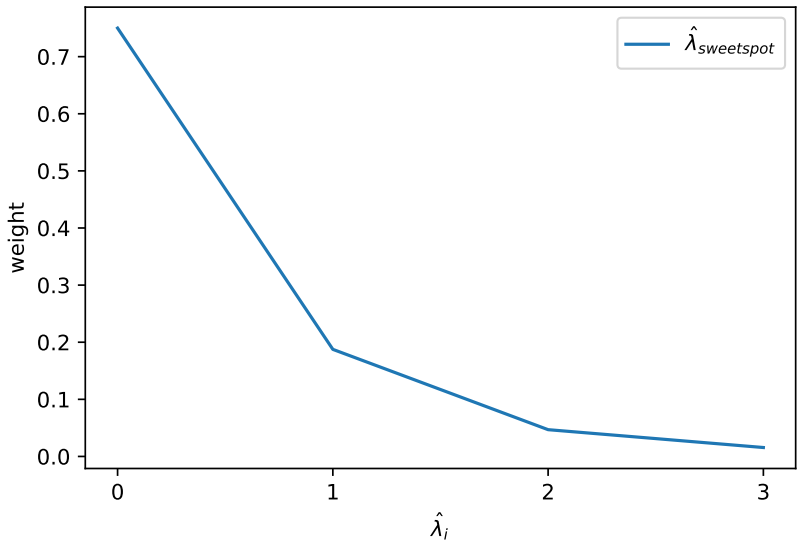


Figure 10: $\hat{\lambda}_{sweetspot}$ that we settle on, after the empirical search, for $N = 4$

4.7 Relationship between mean test accuracy and N

In this part we will observe the performance of the three factories defined in Equations 4, 5 and 6. To provide the reader with visual examples, we provide Figures 11 and 12 resulting from *Fixed-mixup* with $\hat{\lambda}_{sweetspot}$ and $\hat{\lambda}_{maxobf}$ for $N = 5$.

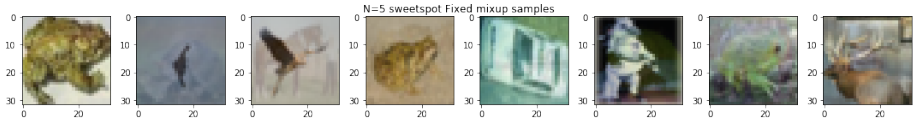


Figure 11: Random CIFAR10 samples mixed with $\hat{\lambda}_{sweetspot}$

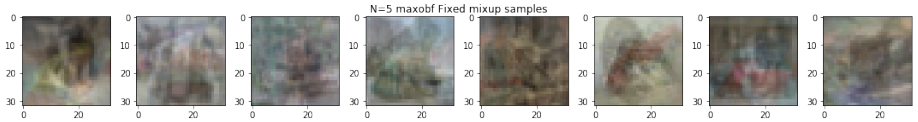


Figure 12: Random CIFAR10 samples mixed with $\hat{\lambda}_{maxobf}$

dataset	model	lambda	N=2	N=3	N=4	N=5
FMNIST	CNN	nomix	90.11	89.56	89.78	-
CIFAR10	ResNet18	nomix	84.74	85.38	85.38	-
CIFAR100	ResNet18	nomix	53.86	55.69	54.93	-
FMNIST	CNN	sweetspot	91.17	91.02	91.05	91.57
CIFAR10	ResNet18	sweetspot	88.3	88.12	88.1	88.76
CIFAR100	ResNet18	sweetspot	58.27	57.93	57.59	57.75
FMNIST	CNN	maxobf	90.88	89.43	88.62	87.05
CIFAR10	ResNet18	maxobf	80.73	80.09	75.36	72.8
CIFAR100	ResNet18	maxobf	51.37	42.54	40.46	31.15

Table 7: Results for *Fixed-mixup* with $\hat{\lambda}_{nomix}$, $\hat{\lambda}_{sweetspot}$ and $\hat{\lambda}_{maxobf}$ as we increase N from 2 to 5

Note that this table provides 3 data for $\hat{\lambda}_{nomix}$ for each dataset+model, but those have no relationship with N (by definition of $\hat{\lambda}_{nomix}$) - they are given to benchmark the other strategies, and to figure out how they compare to no mixup at all.

We provide visual representation of this data in Figure 13, note that the dotted red line indicates benchmark mean test accuracy averaged over 3 trials. From those results we can draw the following conclusions from our two lambdas:

- As N increases, we observe a general drop in test accuracy for the *maxobf* lambda. Nonetheless, the test accuracy stays very impressive considering the obfuscation of the images, presented in Figure 12, i.e compared to the large difference in obfuscation, we observe a small difference in performance. This property of *Fixed-mixup* could be used to obfuscate training samples.

- As N increases, we observe no drop in test accuracy for the *sweetspot* lambda. Since the first image weighs 0.75 in all cases, the test accuracy remains approximately constant. We can imagine this holds true for all N. We also confirm that *Fixed-mixup* provides test accuracy improvement accross models and datasets.

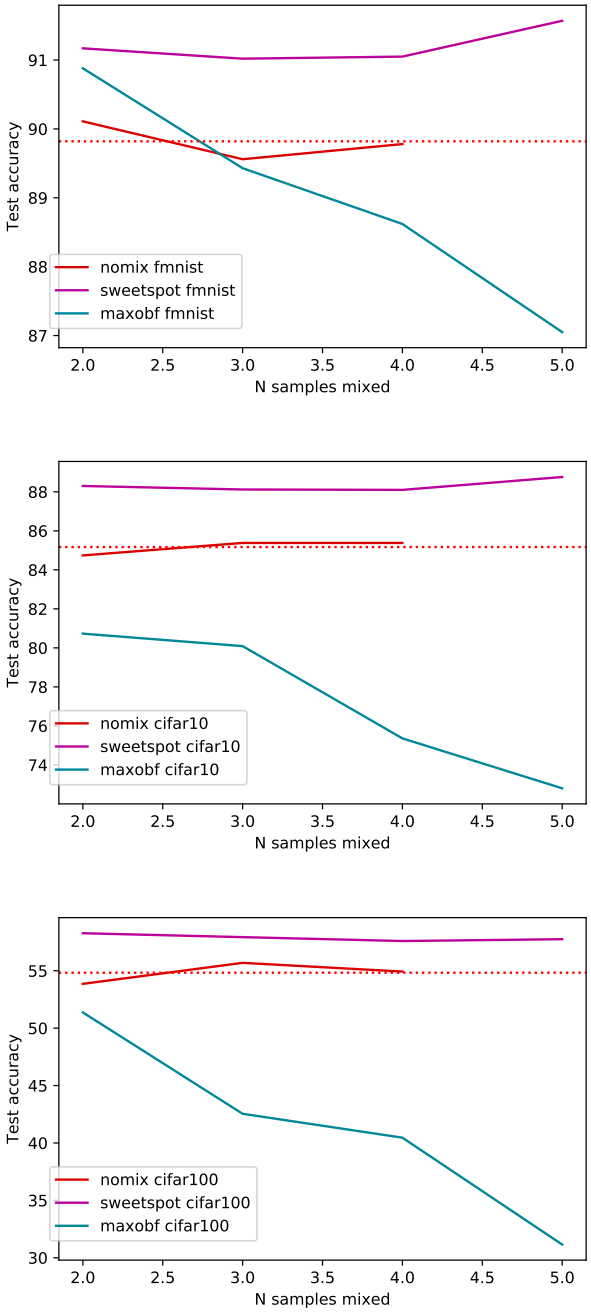


Figure 13: Comparaison of test accuracy of lambdas as N increases

4.8 Fixed-mixup insights

To summarize this section, we confirmed the performance increase of *mixup* held under *Fixed-mixup*.

We introduced the generalized *Fixed-mixup* allowing for the privacy level to be increased with $N \uparrow$.

We introduced the concept of a *sweetspot* $\hat{\lambda}_{sweetspot}$ searched within the range of possible $\hat{\lambda}$, and provided an expression empirically shown to boost test accuracy for generalized *Fixed-mixup* up to $N = 5$ for several models and datasets. We also provided results for $\hat{\lambda}_{maxobf}$, to provide the range of freedom between the two to balance between obfuscation and performance.

We put forward the generalized *Fixed-mixup* as an obfuscation technique for our protocol in Figure 1, with both parameters N and $\hat{\lambda}$ adjustable to gain performance or obfuscation according to the needs.

5 Exploring Pixel-Shuffling

5.1 Pixel-shuffling description

We will present the results of training a model on samples with pixels shuffled by a fixed shuffling mask S. We use a single mask instead of a mask per class to avoid the scenario that the neural network learns about the mask to classify instead of the pixels themselves.

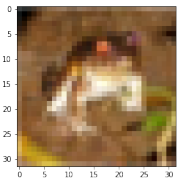
Consider our images dataset $D : \{x_i\}$ where x_i is a row vector of length $s = width \times height$ The mask is created first by generating a permutation P of the elements of the pixel indexation $[0, s)$.

The virtual samples are then generated as follows:

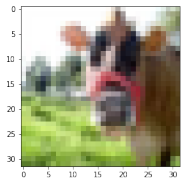
$$\tilde{x_i} : \tilde{x_{i,j}} = x_{i,P(j)}$$

where j indexes the pixel in the row vector image.

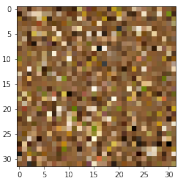
The implementation is straight forwards, and the effect of the procedure is shown in Figure 14.



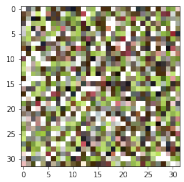
(a) CIFAR10 image before shuffling



(b) CIFAR100 image before shuffling



(c) CIFAR10 image after shuffling



(d) CIFAR100 image after shuffling

Figure 14: Pixel-shuffling

5.2 Pixel-shuffling usefulness

Pixel-shuffling has been discussed as a way to encrypt images, by modifying the **position** information of pixels instead of the **color** information, with techniques such as Chaos-based Pixel-shuffling, useful in certain environments such as wireless communication, when bandwidth utilization should be spared [9].

Pixel shuffling is irreversible without knowing the shuffling indexation used. The nature of unshuffling provides possible set of groundtruth G to the order of :

$$G : |G| = (w \times h)!$$

As image resolution increases, possible groundtruths increases in a combinatorially explosive manner, and multiple plausible images could be decrypted without information on which was plausibly encrypted - the information of pixel locality is lost.

Still, it has been shown that CNN training is somewhat robust to Pixel-shuffling for tasks such as classification. This could provide us with a very good method to apply at the *obfuscate* step in our protocol illustrated in Figure 1 [10].

A fully connected neural network performance would be invariant to the shuffling, however in our models the top performance is capped at a lower level because shuffling destroys locality of features, the principle around which convolutional neural networks are designed.

We seek to answer:

- Can we replicate the results of non-negligible performance of CNNs on

Pixel-shuffled images seen in [10] on our setups, i.e Would a model perform better than randomly guessing when trained on Pixel-shuffled images?

- How does applying *Fixed-mixup* affect Pixel-shuffled images?
  - Would *sweetspot* increase performance while preserving obfuscation?
  - Would *maxobf* preserve performance while increasing obfuscation?

Note: Using a fixed shuffling for images, we could choose between a shuffling mask for every class or a shuffling mask for each class. However, the classwise shuffling could leak information about the class to the model, where it could potentially learn the shuffling to classify instead of using pixel values themselves. By fixing the same mask to each class, we obfuscate the samples while forcing the model to use the pixels only to classify samples.

5.3 Pixel-shuffling benchmark

To benchmark our trials with Pixel-shuffling and *Fixed-mixup*, we initially train our classical setups with a simple shuffling. Visual representation of resulting samples can be seen in Figure 14.

We perform four experiments on each setup and compute the mean test accuracy, and present the results in Table 8.

	CIFAR10	CIFAR100
Pixel-shuffling 0	55.38	27.00
Pixel-shuffling 1	56.05	27.35
Pixel-shuffling 2	56.32	27.42
Pixel-shuffling 3	56.14	27.47
Mean	55.9725	27.31

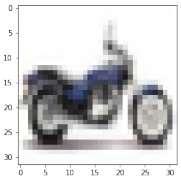
Table 8: Four experiments on the CIFAR datasets and resulting mean test accuracy

We conclude that our results with CIFAR10 are consistent with those presented in [10], at around 55% test accuracy. The drop in performance is the price paid to gain an irreversible visual obfuscation of the samples.

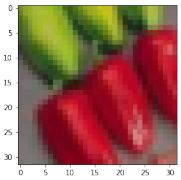
5.4 Pixel-shuffling+*Fixed-mixup*

Now that we benchmarked our models, we are going to apply *Fixed-mixup a posteri* and observe the results. We want to apply *Fixed-mixup* to see if destroying locality of features cancels the effect of the technique discussed in Section 4, or if *Fixed-mixup* still boosts the test accuracy of classical CNNs. Would the latter be true, we could use *Fixed-mixup* to both obfuscate the color values after Pixel-shuffling, and increase the performance to compensate for the drop caused by Pixel-shuffling.

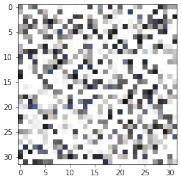
The method is illustrated in Figure 15



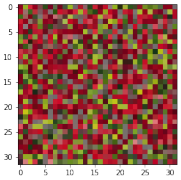
(a) Ingredient A



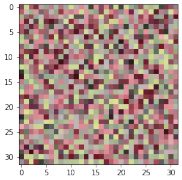
(b) Ingredient B



(c) Shuffled A



(d) Shuffled B



(e) Resulting sample

Figure 15: Pixel-shuffling+*Fixed-mixup* process with $N = 2$ and $\hat{\lambda}_{maxobf}$

We run Pixel-shuffling+*Fixed-mixup* with $N \in [2, 5]$, report the results in Table 9, and illustrate the results in Figure 16.

	2	3	4	5
maxobf cifar10	55.91	55.15	51.21	48.04
sweetspot cifar10	57.68	56.22	56.67	55.73
maxobf cifar100	29.46	25.95	25.85	22.49
sweetspot cifar100	29.54	28.75	29.44	28.56

Table 9: Test accuracy of Pixel-shuffling+*Fixed-mixup* for $N \in [2, 5]$

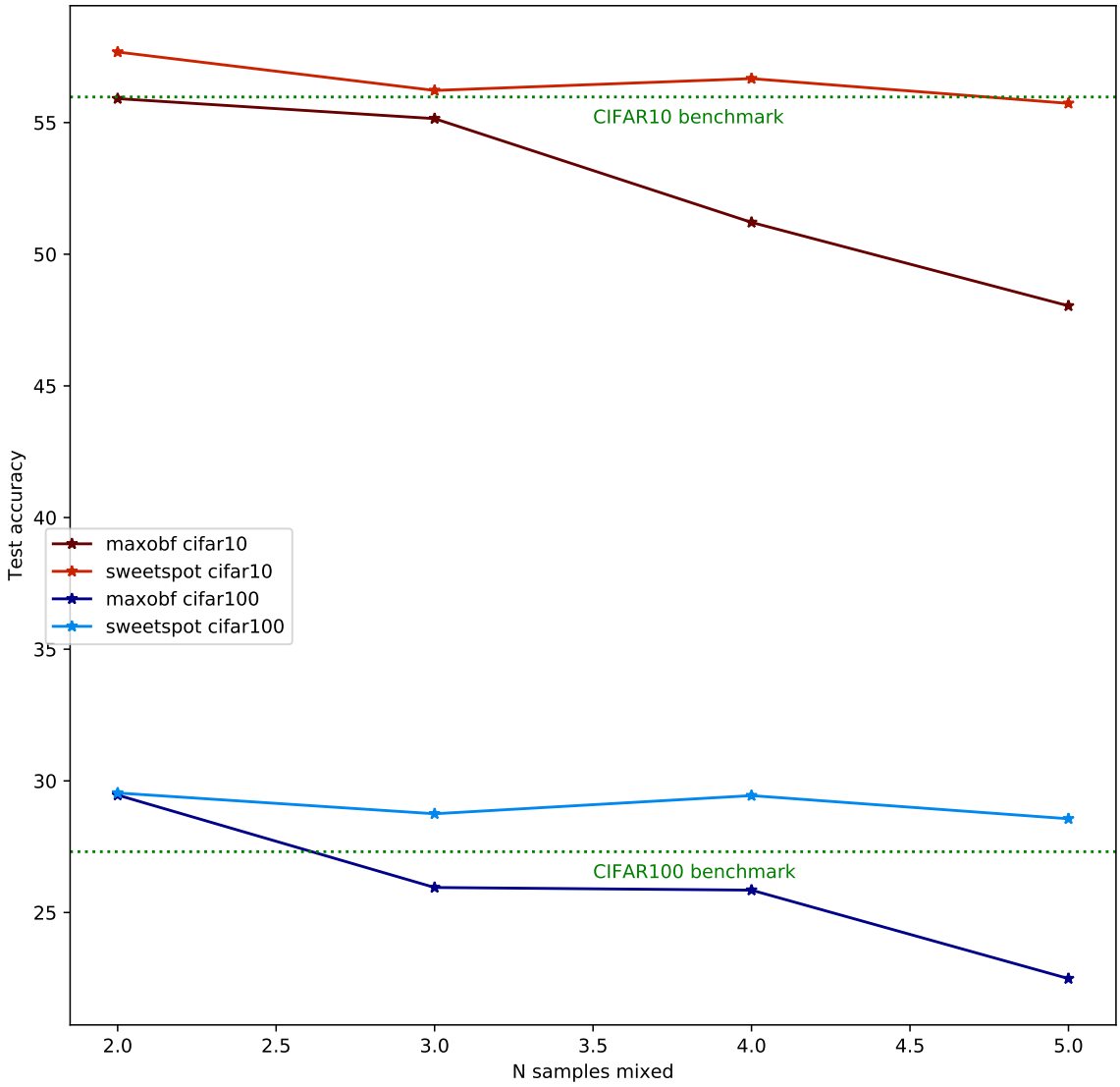


Figure 16: Performance of Pixel-shuffling+*Fixed-mixup* as N increases compared to benchmark values

It appears that *Fixed-mixup* presents the same effect on Pixel-shuffled samples similar to classical *Fixed-mixup* despite the destruction of the locality of features, even-though more trials are needed to confirm statistical significance. $\hat{\lambda}_{maxobf}$ similar to vanilla *Fixed-mixup* presents a decrease in performance as $N \uparrow$, where $\hat{\lambda}_{sweetspot}$ doesn't.

5.5 Pixel-shuffling insights

We confirmed the effects seen in [10], and suggest that *Fixed-mixup* is a method to increase the obfuscation and to provide a certain amount of differential privacy by obfuscating the colors of the Pixel-shuffled images, and there is some degree of freedom in this choice.

Fixed-mixup, if only the exact color value should be obfuscated but close colors are not a problem, can increase the performance of CNN training on Pixel-shuffled images and slightly obfuscate the true color data by using weights $\hat{\lambda}_{sweetspot}$.

However, if colors should be as far away from the originals, using weights $\hat{\lambda}_{maxobf}$ will provide a good color obfuscation at a limited performance price.

For small values of N, our results show that it is possible to increase privacy with this method without negatively affecting performance, offering a no-compromise advantage over the alternative of Pixel-shuffling only.

6 Exploring Pixel-Grafting

6.1 Pixel-Grafting description

We introduce a method called *Pixel-Grafting*, where pixels from one sample are grafted onto another sample at proportion p to create one virtual training sample.

The intuition between this method is that, like *Fixed-Mixup*, *Pixel-Grafting* combines several samples in one image at different proportion to mimic the effect of the original method.

Consider our original images x_i sized $width \times height = s$ represented as column vectors of $R^{s \times 1}$.

We can imagine a pixel-selecting row vector (or *mask*)

$$M \in R^{s \times s} : M_{i,i} \sim B(1, p)$$

picked according to a Bernoulli distribution of parameter p, and we construct our virtual samples via:

$$\tilde{x} = Mx_i + (I_{s\times s} - M)x_j$$

where $J_{s\times 1}$ is a matrix of ones.

The empirical vicinal risk of this method would be noted:

$$R_v(f) = \frac{1}{m}\sum_{i=1}^{m} l(f(\tilde{x_i}), \tilde{y_i})$$

with virtual samples constructed accordingly to the method described above.

The implementation of this method yields images shown in Figure 17.

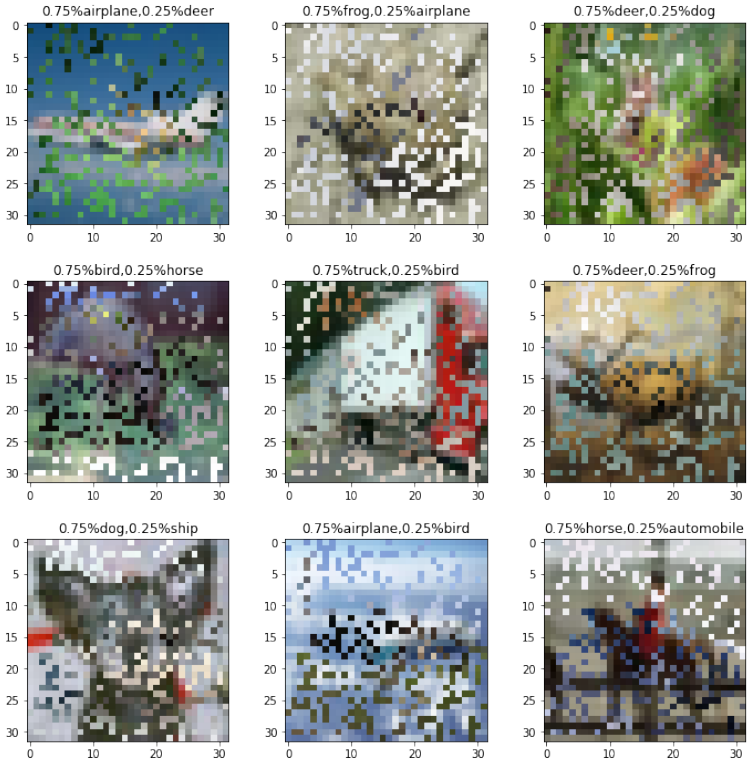


Figure 17: Resulting CIFAR10 samples using pixel grafting with $p = 0.75$

The method should be generalizable to inter-graft $N > 2$ samples, with $\hat{p} = [p_0...p_N]$ respecting constraints 2 and 3 analog to $\hat{\lambda}$ in *Fixed-mixup*, however this experiment was only conducted at $N = 2$.

6.2 Pixel-Grafting usefulness

Pixel-grafting could be viewed as a stochastic, discrete method to emulate the principle of *Fixed-mixup*. Pixel-grafting does not need the full pixels of both samples like *Fixed-mixup* does, and therefore could be used to train a neural network with partial image information , for example if some data of an image gets corrupted, essentially "healing" an image by grafting.

Some additional notes about *Pixel-grafting*:

- The results for *Pixel-grafting* at $p = 1$ are equivalent to the Benchmark laid out in Table 1.
- In our implementation the same mask is used everywhere to avoid leaking classification information through the mask

We want to answer:

- Would a model perform better than with original samples using grafting, or the perform similarly while providing obfuscation, similar to the effect observed in Section 4?
- How does Pixel-Grafting perform compared to *Fixed-mixup*, using graft probability p in *Fixed-mixup* with $\hat{\lambda} = [p, 1 - p]$?

6.3 Pixel-grafting results

As noted in Table 1, the benchmark for the CIFAR10 and CIFAR100 datasets was a mean test accuracy of resp. 85.17 and 54.83.

	0	1	2	3	4	mean
0.875	85.11	85.67	85.09	-	-	85.29
0.75	83.47	84.61	83.54	84.57	83.86	84.01
0.625	80.11	80.89	82.46	-	-	81.153333
0.5	78.76	78.35	80.80	79.57	77.95	79.086

Table 10: Results of *Pixel-grafting* for CIFAR10+ResNet18

	0	1	2	3	mean
0.875	56.21	56.66	55.04	-	55.97
0.75	53.14	54.03	52.58	52.40	53.0375
0.625	50.68	49.94	50.57	-	50.3967
0.5	49.58	49.03	49.14	50.09	49.46

Table 11: Results of *Pixel-grafting* for CIFAR100+ResNet18

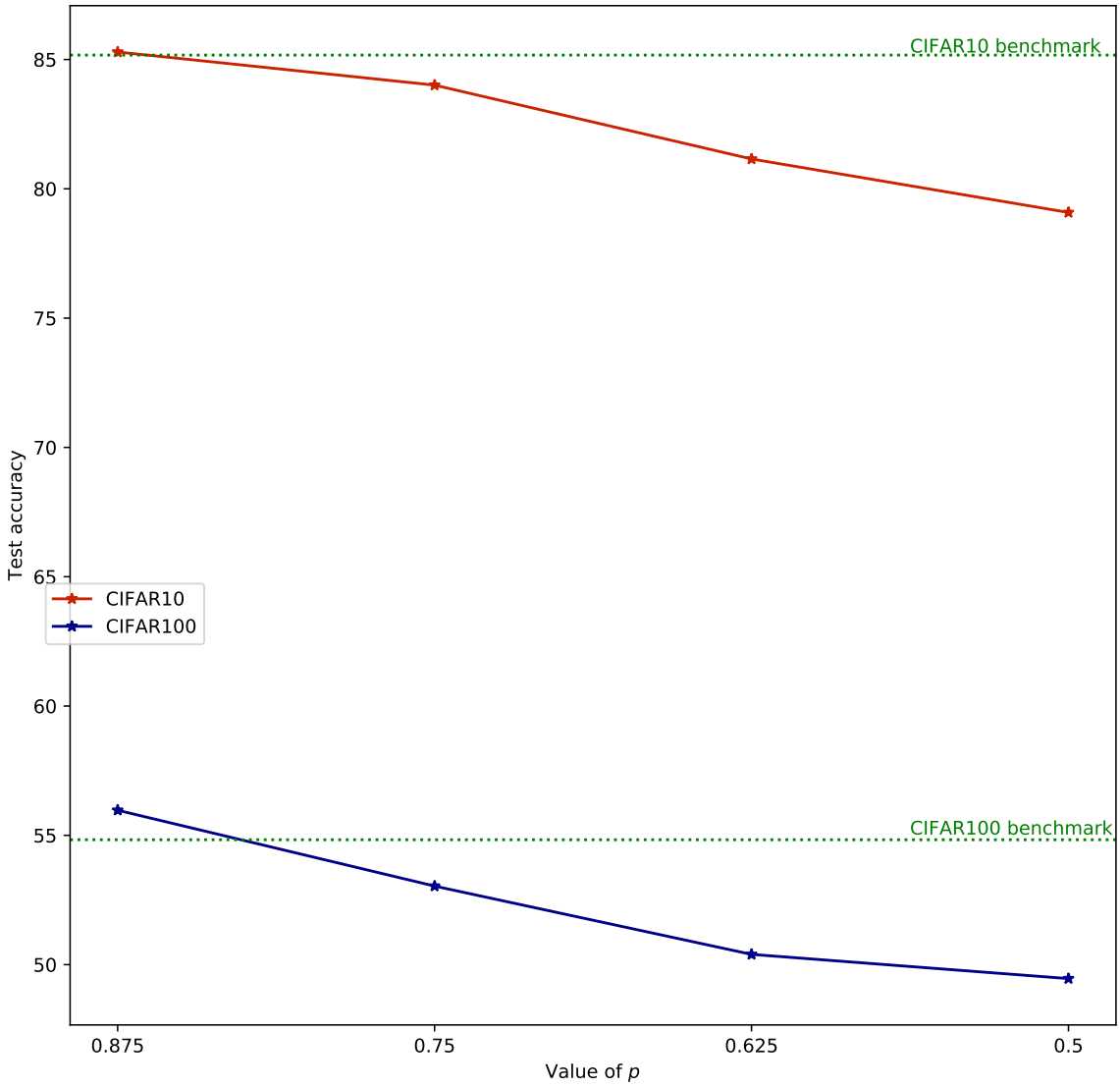


Figure 18: Performance of *Pixel-grafting* compared to benchmarks

6.4 Pixel-grafting insights

The following insights could be drawn from those results:

- *Pixel-grafting* outperformed the benchmark slightly at $p = 0.875$, averaged over 3 experiments both. The difference is too slim to affirm statistical significance. *Pixel-grafting* still under-performed *Fixed-mixup* at equal ratios.
- *Pixel-grafting* does not exhibit the "sweetspot" behavior that *Fixed-mixup* showcases, or it is at $p > 0.875$
- *Pixel-grafting* is mostly robust to performance deterioration at $p_{maxobf} = 0.5$. This could provide a safer alternative to *Fixed-mixup* for $N = 2$ in order to render some type of sensitive images un-readable, such as license plate or face images. More trials would need to be done, but a mix of N-variate *Pixel-grafting* and *Fixed-mixup* could provide strong visual obfuscation/privacy in performant training data.

7 Conclusion

At first, we have confirmed the *mixup* performance increase in a modified version that we named *Fixed-mixup*, introduced a generalized technique for any N, and presented the results of applying this technique on classical deep learning models and datasets for several N. We also searched the weight space and proposed an expression for a $\hat{\lambda}_{sweetspot}$ weight vector which, within our search space, were the optimal weights to increase performance, a result which could be used directly to increase neural network performance. We also provided results for $\hat{\lambda}_{maxobf}$ for multiple models and datasets, to analyse how N relates with performance in the context of image obfuscation. This leaves the obfuscation user the choice on how to mix performance and obfuscation for the results and privacy needed.

We explored another image obfuscation technique, Pixel-shuffling, reported benchmark results confirming previous research, and reported results when applying *Fixed-mixup* to the technique. We have shown that the *Fixed-mixup* technique increases Pixel-shuffling-trained convolutional neural networks performance at $\hat{\lambda}_{sweetspot}$ and increases obfuscation at $\hat{\lambda}_{maxobf}$ without negatively impacting performance at least for small N, which could be directly used in applications where Pixel-shuffling is traditionally needed.

Finally, we presented and discussed the potential of a *Fixed-mixup*-inspired data mixing technique called *Pixel-grafting*, presented results, and showed that there might not be a significant performance increase as there exists with *mixup*, however we showed that it remains available as an alternative technique for data obfuscation.

The project as a whole should provide the reader with an extensive array of results and leads for image obfuscation with the goal of performing privacy-preserving Machine Learning.

References

[1] Y. N. D. D. L.-P. Hongyi Zhang, Moustapha Cisse, "mixup: Beyond empirical risk minimization," *International Conference on Learning Representations*, 2018.

[2] K. Lee, H. Kim, K. Lee, C. Suh, and K. Ramchandran, "Synthesizing differentially private datasets using random mixing," in *2019 IEEE International Symposium on Information Theory (ISIT)*, pp. 542–546, IEEE, 2019.

[3] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *et al.*, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

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

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

[6] A. Krizhevsky, "Learning multiple layers of features from tiny images," tech. rep.

[7] O. Chapelle, J. Weston, L. Bottou, and V. Vapnik, "Vicinal risk minimization," in *Advances in neural information processing systems*, pp. 416–422, 2001.

[8] A. Hyvärinen and E. Oja, "Independent component analysis: algorithms and applications," *Neural networks*, vol. 13, no. 4-5, pp. 411–430, 2000.

[9] M. Prasad and K. Sudha, "Chaos image encryption using pixel shuffling," *CCSEA*, pp. 169–179, 2011.

[10] C. Ivan, "Convolutional neural networks on randomized data," *arXiv preprint arXiv:1907.10935*, 2019.