Direct Neural Network Training on Securely Encoded Datasets

Anonymous authors
Paper under double-blind review

Abstract

In fields where data privacy and secrecy are critical, such as healthcare and business intelligence, security concerns have limited data availability for neural network training. A recently developed technique securely encodes training, test, and inference examples with an aggregate non-orthogonal and nonlinear transformation that consists of steps of padding, perturbation, and orthogonal transformation, enabling artificial neural network (ANN) training and inference directly on encoded datasets. Here, the performance characteristics of the various aspects of the method are presented. The individual transformations of the method, when applied alone, do not significantly reduce validation accuracy. Training on datasets transformed by sequential padding, perturbation, and orthogonal transformation results in only slightly lower validation accuracies than those seen with unmodified control datasets (relative decreases in accuracy of 0.15% to 0.35%), with no difference in training time seen between transformed and control datasets. The presented methods have broad implications for machine learning in fields requiring data security.

Keywords: secure machine learning, data security, encoded training data, encoded test data, non-orthogonal transformation, nonlinear transformation, artificial neural networks.

1 Introduction

Privacy and data security are paramount in areas such as healthcare and business intelligence. In healthcare, as an example, complex regulatory structures exist in different jurisdictions that were designed to protect the privacy of data that might identify an individual and their personal information (Theodos & Sittig, 2020; Phillips, 2018). The data security requirements in these areas have created demand for new technologies that can securely encode data for machine learning training and inference using cloud resources.

Massive amounts of data remain siloed in hospital and other systems because of privacy, data security, and regulatory concerns (Kostkova et al., 2016), and these data remain inaccessible to the cloud resources and technical expertise needed for modern neural network training. The existing barriers to cloud-based machine learning and collaboration using this wealth of healthcare data are particularly important in light of recent dramatic results that have been obtained by training neural networks on the limited number of datasets that have been available to date. For example, deep network training on retinal photographs from diabetic screening programs yielded an accuracy for the detection of diabetic retinopathy that is on par with board-certified ophthalmologists (Gulshan et al., 2016). In addition, use of the same retinal imaging datasets facilitated training of neural networks to make a range of surprisingly accurate inferences related to blood pressure, smoking history, age, and gender (Poplin et al., 2018). The same group more recently trained neural networks on external eye photographs, and these networks could accurately predict levels of the laboratory measures hemoglobin A1C (a measure of diabetes mellitus and its control) and serum cholesterol (Babenko et al., 2022), neither of which were previously known to be associated with any features visible in external photographs of the eye.

There should be an important role for neural networks to dramatically enhance clinical diagnosis (Poplin et al., 2018; Babenko et al., 2022), but machine learning techniques have yet to be widely adopted across healthcare. While the slow pace of adoption may be in part related to a slow-to-adapt culture in healthcare and a complex regulatory environment, data siloing has certainly limited the availability of data and created barriers to transmitting private data to cloud resources for training and inference. Solutions are therefore needed to securely encode data in healthcare and other fields such that encoded data may be directly used in machine learning applications.

An ideal encoding method to establish security for machine learning systems would securely encode training, test, and inference data in a fashion that allows existing architectures to be used on the encoded data, with minimal loss of accuracy and comparable training times. Such an implementation would also require test and inference examples to be similarly encoded to training examples, which would have the benefit of maintaining the business value of training data in collaborations.

Here, the performance of a novel method of secure data encoding for direct use with neural networks (Anonymized, 2022) is described and tested. The method involves a combination of different forms of padding, perturbation, and orthogonal transformation (e.g., fixed shuffling) to create an aggregate non-orthogonal and nonlinear transformation that is applied to training, test, and inference data, with neural networks trained directly on the encoded data.

2 Description of the encoding methods

The central method of the secure encoding approach presented here is an aggregate non-orthogonal and nonlinear transformation comprised of one or more non-orthogonal and nonlinear transformations together with one or more orthogonal transformations. At least one of the included transformations is an orthogonal transformation, which can use a random orthogonal matrix of appropriate size, or can apply an example-wise fixed shuffling (index shuffling) transformation, which represents a subset of the broader set of random orthogonal matrix transformations. For a tensor of rank N, sequential transformations are applied to the input tensor, including at least one step that shuffles each of the examples in a fixed fashion using a stored shuffled index array (or at least one step that dots a random orthogonal matrix of appropriate size with the input vectors). For simplicity of illustration, the specific type of orthogonal transformation illustrated in this paper will be fixed shuffling (index shuffling) unless indicated otherwise. All tensors in a given project's training and test sets, as well as any inference examples, are transformed in similar fashion.

To illustrate the method, three different forms of manipulation, which may be applied in varying combinations, are described below: *padding*, *perturbation*, and *random orthogonal transformation*. While each of these transformations may be applied to tensors of arbitrary rank with any data type suitable for any type of machine learning operation, in this paper, method illustrations and performance experiments will use supervised artificial neural network (ANN) training on the MNIST dataset of grayscale (n,28,28) handwritten digits (Deng, 2012) as an exemplar.

2.1 Padding

Padding of a dataset involves generating a pad of data such that the examples of the source tensor are increased in size and the resulting examples are of uniform shape across the resulting tensor. The pad may represent one or more rows of data, or one or more columns of data, or a combination of one or more rows and columns (as shown in the L-shaped pad in Figure 1B). Any addition of padding pixels outside or inside the image, including insertion of pixels or groups of pixels within the image, is allowed in the method, as long as the resulting examples are of uniform shape across the resulting tensor. Different forms of padding may be used that vary in the way they are applied to the input tensor. Three variations on padding will be described here and performance-tested below: *fixed padding*, *random non-fixed padding*, and *adjusted non-fixed padding*.

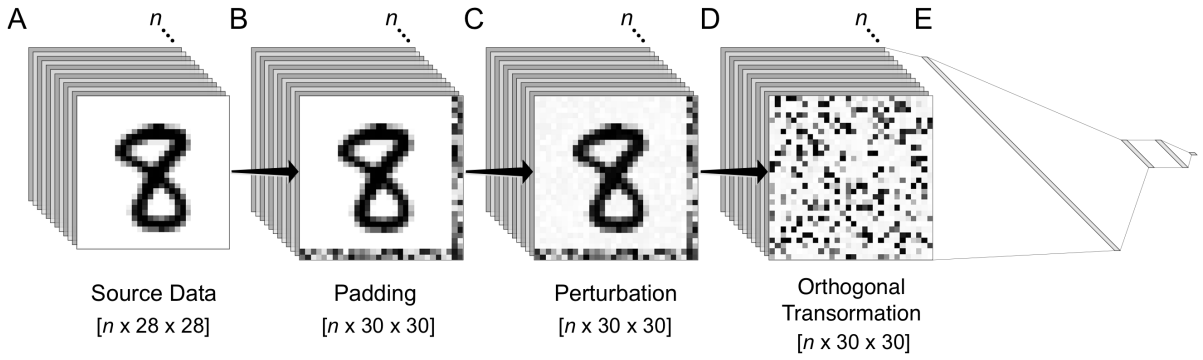

Figure 1: *Padding*, *perturbation*, and *orthogonal transformation* performed serially prior to training of an artificial neural network (ANN).
A. Illustration of the source data, a (n,28,28) tensor, the MNIST handwritten digits dataset with n examples, with an illustrated example showing a handwritten digit corresponding to the number 8.
B. Application of a *padding* transformation. In this case, an L-shaped pad of randomly chosen pixels is generated with 2 rows and 2 columns. This can be *fixed padding* across the examples of the tensor (one randomly generated pad is appended in identical fashion to every example), or it can vary across the examples (*non-fixed padding*: a different pad is appended to each of the examples). Padding can also represent separate addition of x rows or y columns or any addition of extrinsic data elements outside or inside of the tensor, as long as the transformed result has a consistent shape. The result in the present example is a transformed tensor of shape (n,30,30).
C. Application of a *perturbation* transformation. A random array of perturbation values (in the illustrated case, from the range of -5% to +5%) is created and used to alter the pixels by a percentage of the original pixel values of the examples of the (n,30,30) tensor. Cases in which pixel values would be taken beyond the minimum or maximum pixel value at a given location can be handled in different ways - in this case, pixels exceeding the minimum or maximum values are clipped to the minimum or maximum value. The effect of the fixed perturbation transformation can be seen in the figure as a fine amount of light gray pixel noise that is most notable when comparing the white pixels in [B] to corresponding positions in [C]. The perturbation transformation may be a *fixed perturbation* (the same set of mathematical perturbations applied to every example) or a *non-fixed perturbation* (different sets of mathematical perturbations applied to each example).
D. Application of a *random orthogonal transformation*. In this case, a fixed index shuffling is applied to shuffle the pixel index positions of each example in identical fashion. More broadly, any random orthogonal transformation may be applied using a random orthogonal matrix of appropriate size (e.g. for a 30x30 input, a [(30x30) x (30x30)] random orthogonal matrix).
E. The transformations applied serially from [B] through [D] are an aggregate *non-orthogonal transformation*, and the result of this transformation is used to train a multilayer ANN.

2.1.1 Fixed padding

With *fixed padding*, a single pad is created for a source example, typically by random choice for each pixel, within the range of pixel values of the source tensor. This same pad is appended in identical fashion to each example across the n examples of the source tensor, with the same pad used for all examples of the training tensor, the test tensor, and any inference examples.

2.1.2 Non-fixed padding

Non-fixed padding is any padding approach that varies from example to example. In one version of non-fixed padding, *random non-fixed padding*, a completely different random pad is generated for each example across the training, test, and inference tensors. In another version of non-fixed padding, *adjusted padding* starts with a fixed pad of appropriate shape that is stored in memory. This stored pad in memory is then adjusted, pixel by pixel, by a random amount in a specified range (by an analogous process to that applied in the perturbation transformation discussed below), and differently adjusted pads are then applied to each example across the training, test, and inference tensors. Other forms of non-fixed padding, not illustrated here, include methods such as generating a group of random pads that is smaller than the number of examples in an input tensor and then using random choice from this pool of random pads to pad the examples in a tensor.

2.2 Perturbation

Perturbation of a dataset involves applying a set of functions that perturb the pixel value at each position in the examples of an input tensor. In the present experiments, the version of this perturbation shown is an alteration of the pixel values by different percentages of the pixel value range.

First, an array of values from a chosen minimum to a chosen maximum (e.g., min = -5% to max = +5%) is created by random choice within these constraints, where the array size is matched to the size of an example in the input tensor. (In the experiments shown in this paper, source data are transformed from 8-bit depth [range 0-255] to 32-bit floating-point precision [range 0.0 to 1.0] to enable graded changes between the original 8-bit values.) For each example in the input tensor, the pixel value at a given location is perturbed by the randomly chosen percentage amount at the corresponding position in the perturbation array.

As pixel values are perturbed upward or downward by this technique, perturbation beyond the relevant minimum and maximum values must be explicitly handled. While many approaches can be used, in this paper, two approaches are tested: *clipping* and *reflection*. Clipping assigns the minimum or maximum value to any pixel values brought beyond these values by the perturbation algorithm. For example, if a pixel value on a float scale from 0.0 to 1.0 started at 0.99 and would be brought to a value of 1.03 by a perturbation, clipping changes the resulting value to 1.0. Reflection brings the value to a corresponding amount within the min/max boundaries. For example, if a pixel value on a float scale from 0.0 to 1.0 started at 0.99 and would be brought to 1.03 by a perturbation, the amount exceeding the maximum value is (1.03 - 1.0 = 0.03) and the assigned value using reflection would be (1.0 - 0.03 = 0.97). Figures 1C and 2B illustrate the effect of applying a -5% to +5% bounded percentage perturbation with clipping to the examples of an input tensor. With very large perturbation settings, it is possible for reflection to occur back and forth between the max and min values. Throughout this paper, the default setting for handling perturbations beyond the maximum and minimum bounds will be clipping unless stated otherwise.

Different forms of perturbation may be applied that vary in their application to the input tensor. Two variations on perturbation will be described here and performance-tested: *fixed perturbation*, in which all examples are modified in similar fashion, and *non-fixed perturbation*, in which each example is modified in a different fashion.

2.2.1 Fixed Perturbation

In *fixed perturbation*, a single perturbation array is generated, and the set of pixel value perturbations using the perturbation array is applied in identical fashion to each of the examples in the input tensor, with the

same fixed perturbation transformation applied to all of the examples of the training tensor and the test tensor, as well as any inference examples.

2.2.2 Non-fixed perturbation

In *non-fixed perturbation*, a different perturbation array is generated and applied to each example. For example, a different set of pixel variation functions may be generated for each example with a minimum to maximum range for the degree of potential perturbation (e.g. min = -5% to max = +5% perturbation of pixel values). With non-fixed perturbation, different perturbation transformation arrays are applied to each of the examples of the training tensor and test tensor, as well as to any inference examples.

2.3 Orthogonal Transformation

Application of an orthogonal matrix transformation of input vectors has been proposed as a method to encode data for use with artificial neural networks (Chen, 2019), because orthogonal matrix transformation maintains vector length and angles between vectors and thus machine learning accuracy should be maintained by this approach. In the proposed method of Chen (2019), a square image file of appropriate size is used as a key that is subjected to QR factorization to yield an orthogonal matrix. The orthogonal matrix thus obtained is used to encode training and test data. Unfortunately, when used in isolation as described, orthogonal matrix transformation suffers from an important security vulnerability common to all square matrix linear transformations (see Discussion below). Therefore, in the methods presented here, the orthogonal matrix transformations applied are used in combination with steps of padding and perturbation to yield an aggregate non-orthogonal and nonlinear transformation that does not suffer from this vulnerability.

Two subtypes of orthogonal transformation are used here in combination with the transformations of padding and perturbation described above. In the broadest method, a *random orthogonal matrix* of appropriate size is generated and used to dot input vectors to obtain encoded vectors. In a specific subtype of random orthogonal matrix transformation, *fixed shuffling* (index shuffling) is applied to input vectors to obtain encoded vectors.

2.3.1 Random orthogonal matrix transformation

To transform input vectors by random orthogonal transformation, a random orthogonal matrix is generated, either by random choice of an orthogonal matrix of appropriate shape drawn from the O(N) Haar distribution, or by initial generation of a random matrix of appropriate shape that is then subjected to QR factorization to obtain a random orthogonal matrix of appropriate shape. Once a random orthogonal matrix is generated by either method, it is stored in memory and used to transform input vectors by taking the vector · matrix dot product of each input vector and the stored random orthogonal matrix. The same random orthogonal matrix is used to transform training, test, and inference data in the same fashion.

2.3.2 Fixed shuffling

Fixed shuffling, or index shuffling, of a dataset involves shuffling the pixel index locations of the examples of an input tensor such that the same shuffling rearrangement is applied in identical fashion to each example in the tensor. Fixed shuffling is a subset of the broader set of random orthogonal matrix transformations. Fixed shuffling can be accomplished by randomly generating a shuffling index array that matches the shape of an example in the input tensor, which stores the instructions for rearranging the pixel locations in each example. For example, a shuffling index array might indicate that the pixel residing at location (0,0) in each example be repositioned to position (16,9) in each example in the fixed shuffled tensor, with a corresponding repositioning of every pixel index location in each example. The identical result may, of course, be obtained by flattening the examples, using a flat shuffling index array of corresponding length, then reshaping the result back to the original (x,y) example shape, and fixed shuffling may be applied to examples of any dimension in tensors of any rank. Alternatively, fixed shuffling may be applied by generating a square orthogonal matrix filled with zeroes except for single ones in unique column positions across the matrix rows, and this matrix may be used to have the same index shuffling (coordinate axis permutation) effect. As described above for the padding and permutation transformations, the same fixed shuffling transformation is applied to the examples of the training tensor and the test tensor, as well as any inference examples. Figure 1D shows the

effect of applying a fixed shuffling transformation to the examples of an input tensor, in this case the result of the fixed padding transformation shown in Figure 1B followed by the fixed perturbation transformation shown in Figure 1C.

2.3.3 Non-fixed shuffling

Unlike the transformations of padding and perturbation described above, *non-fixed shuffling* (in which a different shuffle is applied to each example of the training and test datasets) is of limited practical utility in the context of training machine learning systems for inference—separate example-wise shuffling disrupts the spatial relationships of the data elements across examples, thus requiring a machine learning system to train on non-spatial data alone. This effect is, however, of academic interest, because it allows for a quantitative assessment of the extent to which a particular machine learning system uses spatial vs. non-spatial information to train. Specifically, training on a fixed shuffled dataset involves both spatial information (e.g., the arrangement of pixels in the original dataset) and non-spatial information (e.g., the percentage of darker pixels in some examples compared with others, which can in turn associate with the example label), while training on a non-fixed shuffled dataset involves only non-spatial information. A comparison between the results of these two treatments (fixed shuffling vs. non-fixed shuffling) thus allows for a quantitative assessment of how these two aspects of a given dataset (spatial vs non-spatial information) contribute to network training.

2.4 Serial combinations of transformations

As shown in Figure 1, the transformations of padding, perturbation, and orthogonal transformation may be performed sequentially, in varying combinations. (The sequential application of multiple transformations can, of course, be established as a single algorithmic transformation that has the same effect). At least one orthogonal transformation step is used to achieve a secure encoding result, and typically an orthogonal transformation step is deployed at the final stage of the chain of transformations, as illustrated in Figure 1. The effect of combining the non-orthogonal and nonlinear transformation steps of padding, perturbation, or a combination of these, is to create an aggregate non-orthogonal and nonlinear transformation.

3 Methods

3.1 Experimental setup

Training performance of the encoding steps of padding, perturbation, and orthogonal transformation, alone and in different combinations, was tested here by training and validation with the MNIST dataset (Deng (2012)), with the usual 60K training and 10K validation split. Training and validation were performed using the same ANN structure and hyperparameters for all trained models, using TensorFlow / Keras in Python 3.6 on Google Colab+, running on a NVIDIA Tesla T4 or P100 GPU, depending on the server connection. All training sessions for transformed datasets and paired controls were performed on the same server connection, using the same available GPU. For the training time comparisons described below, a server connection running a NVIDIA Tesla T4 GPU was used. ANN structure was (1) an input layer corresponding to the example data shape with or without change in shape produced by encoding, (2) two fully-connected hidden layers of 128 neurons each with ReLU activation, with 0.25 Dropout, Flatten, and 0.5 Dropout, and (3) an output layer of 10 neurons corresponding to the 0-9 digit labels of MNIST, with Softmax activation. The loss function was categorical crossentropy, and the optimizer was RMSprop, with a learning rate of 0.001, rho of 0.9, and epsilon of 1e-07. Training batch size was 128, with 12 training epochs for all training runs. To facilitate appropriate comparisons of validation accuracy and of training time, model parameters were fixed for all experiments and not individually optimized—the only difference from model to model was any required increase in input shape produced by the encoding process (specifically the increase in input shape produced by padding). To achieve appropriate precision of fixed perturbation transformations, all pixel values for all models were converted to floating point precision.

Each encoding process that was tested, whether individual encoding steps or multiple encoding steps in different combinations, was performed a total of 15 separate times (see *sample size calculations* below),

with separate ANN models trained for each from scratch. For example, to test orthogonal transformation alone, as shown in Figure 2C, a new random orthogonal matrix was generated to transform the training and validation splits of MNIST, and then the model underwent training. This was repeated 14 more times, each time deleting all model variables, generating a new random orthogonal matrix, transforming the tensors *de novo*, and training a new model from scratch. During the same session, while connected to the same Google Colab+ cloud server with the same available GPU, control models were independently trained using MNIST without any encoding steps. For each comparison, 15 control models were trained, with presented controls paired to the same server session used for the encoding process being tested, to ensure similar conditions (such as available GPU) and to avoid multiple comparisons in statistical analysis.

3.2 Statistics and sample size calculation

For comparison of validation accuracy between controls and a given encoding process, the nonparametric Kruskal-Wallis equality-of-populations rank test was performed, with P value for the Chi-square test without ties reported for each comparison (similar results were obtained with the parametric Student's t-test).

Sample size power analysis was performed for two-sample means using a series of 20 control runs (unencoded MNIST) on the same ANN, which showed a control mean validation accuracy of 0.977 and standard deviation of the validation accuracy of 0.001, yielding a very small sample size because of the extremely tight distribution of results reflected in the very small standard deviation. Conservatively assuming a standard deviation 50x larger than this (0.05) for both groups yields a sample size estimation of 10 model trainings needed per group (control vs. encoded) to detect a difference in accuracy of 0.007 (0.977 for control, 0.970 for encoded) with power of 0.8 and alpha of 0.05. Keeping the very low observed standard deviation for the control group (0.001) but conservatively assuming a 50x higher standard deviation for the encoded group (0.05) for the same detection of a 0.007 difference in accuracy yields a sample size estimation of 7 model trainings per group. Based on the above calculations, a conservative choice of 15 training runs per group was used for experiments comparing performance between control and encoded data, and for training time experiments, 20 training runs per group was used. For training time experiments, elapsed time was measured from the start of the first training epoch to the end of the last training epoch.

Statistical tests and sample size calculations were performed with Stata/MP 16.1 (StataCorp, College Station, TX). Illustrations were generated using the Python library Matplotlib in Google Colab+, and graphs were generated in GraphPad 8.4 (GraphPad Software, San Diego, CA).

4 Results

4.1 Single transformations

Figure 2 shows model training performance for each of the transformations described above (padding, perturbation, and orthogonal transformation) applied as separate, single-step transformations and compared with training with unencoded MNIST (control). For padding transformation (Figure $2A_{1\text{-}3}$), the results of fixed padding (application of the same random pad to every example) are shown in Figure $2A_2$, and the results of non-fixed padding (application of a different random pad to every example) are shown in Figure $2A_3$. For perturbation transformation (Figure $2B_{1\text{-}3}$), the results of fixed perturbation (application of the same set of perturbation transformations to every example) are shown in Figure $2B_2$, and the results of non-fixed perturbation (application of a different set of perturbation transformations to every example) are shown in Figure $2B_3$. For orthogonal transformation (Figure $2C_{1\text{-}3}$), results of random orthogonal matrix transformation are shown in Figure $2C_2$, and the results of fixed shuffling are shown in Figure $2C_3$. For each of the transformations applied individually, no significant difference in the validation accuracy was detected between models trained on transformed data vs. control (Figure 2).

4.1.1 Non-fixed vs. fixed shuffling

As discussed above, non-fixed shuffling (application of a different index shuffle to each example across the training and test tensors) is not generally of practical use, as this manipulation disrupts the spatial relation-

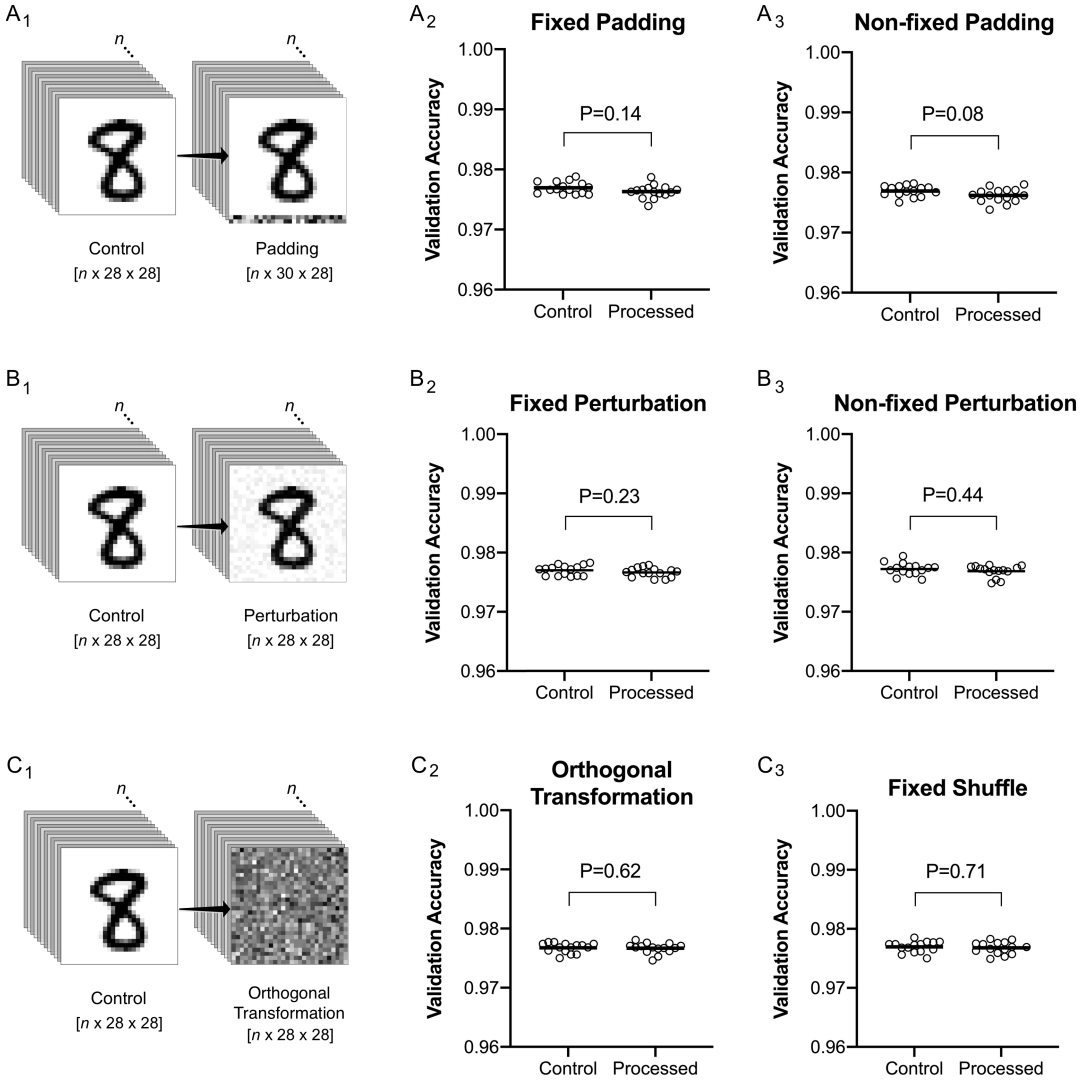

Figure 2: Separate transformations compared with controls.
A_1. Illustration of *padding* (a 2-row random pad appended to the MNIST examples).
A_2. Model training on *fixed padded* data (same random pad for every example). Mean validation accuracy was 0.9763 ± 0.001 for padded data (Processed), and 0.9770 ± 0.001 for control (P=0.14).
A_3. Model training on *non-fixed padded* data (a different random pad for each example). Mean validation accuracy was 0.9762 ± 0.0012 compared with 0.9769 ± 0.0009 for control (P=0.08).
B_1. Illustration of *perturbation* (applying a random perturbation array, constrained from -5% to +5% to the examples of MNIST).
B_2. Model training after *fixed perturbation* (same perturbation array applied to every example). Mean validation accuracy was 0.9767 ± 0.0008 (Processed) compared with 0.9770 ± 0.0009 (Control) (P=0.23).
B_3. Model training after *non-fixed perturbation* (a different array of random perturbations applied to each example). Mean validation accuracy was 0.9768 ± 0.001 (Processed) compared with 0.9772 ± 0.001 (Control) (P=0.44).
C_1. Illustration of *random orthogonal transformation*. A single random orthogonal matrix is used to transform every example.
C_2. Model training after *random orthogonal matrix transformation* (as illustrated in C_1). Mean validation accuracy was 0.9767 ± 0.0009 (Processed) compared with 0.9767 ± 0.0008 (Control) (P=0.62).
C_3. Model training after *fixed shuffling*, a subset of the broader set of random orthogonal transformations. Mean validation accuracy was 0.9768 ± 0.001 (Processed) compared with 0.9769 ± 0.0009 (Control) (P=0.71).

ships between data elements (pixels) and thus forces the network to train only on non-spatial information. This effect does, however, allow for an assessment of the extent to which a machine learning system uses spatial vs. non-spatial information to train. Comparing network training between non-fixed shuffled and fixed shuffled MNIST showed that the non-fixed shuffled data (lacking spatial information) yielded a mean validation accuracy of 0.2608 ± 0.0049 compared with 0.9769 ± 0.001 for fixed shuffled controls ($P<0.001$).

4.2 Variation in padding size

The size of the appended pad is one of several adjustable parameters in the methods tested here, so the impact of varying the size of the appended pad on validation accuracy was tested for both fixed and non-fixed padding. Figure 3 shows the results of increasing pad size from 0 rows (control) to 10 rows of padding, with examples illustrated in Figure 3A. For both fixed padding and non-fixed padding, increasing size of the appended pad did not reduce validation accuracy (Figure 3B).

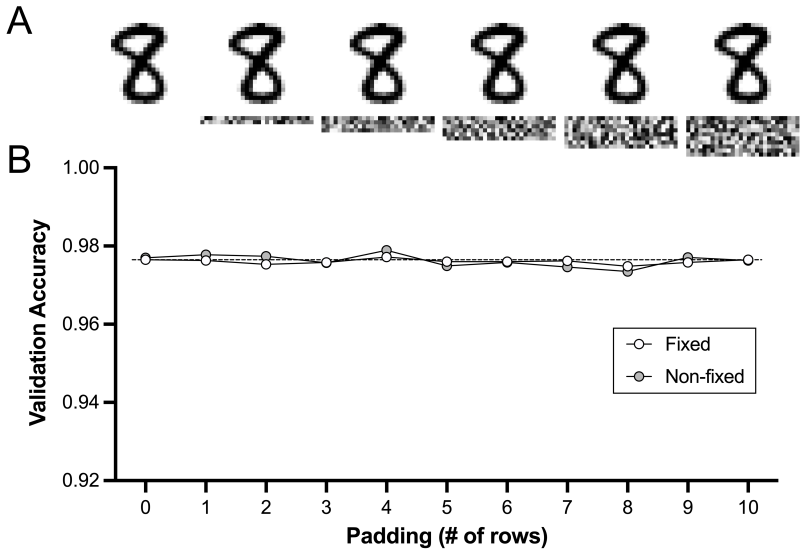

Figure 3: Variation of padding size.
A. Illustration of increasing pad size from 0 pad rows on the left (control) up to 10 pad rows on the right.
B. Validation accuracy of models trained with a single-step transformation using different numbers of pad rows, from 0 to 10 pad rows. White circles show the results of varying pad size using *fixed padding* (the same random pad applied to every example) and gray circles show the results of varying pad size using *non-fixed padding* (a different random pad applied to each example). Horizontal dotted line indicates the baseline from control.

4.3 Variation in perturbation range

Another adjustable parameter in the present methods is the range of potential perturbation of pixels in the examples. In the above illustrations (Figures 1C and 2B1), a perturbation range of -5% to +5% is shown, but this range is arbitrary and can be varied across smaller or larger ranges, creating different degrees of distortion for human viewers depending on the range choice. While only small ranges of perturbation would typically be desirable to achieve security goals together with padding and orthogonal transformation, it is important to establish how varying degrees of perturbation impact model training.

As discussed above, pixel perturbation beyond the bounds of the available pixel values may be handled in several different ways, including clipping (setting the value to the boundary value) and reflection (reflecting the value about the boundary value), and, as expected, these two different approaches yield visibly different results when very large perturbation ranges are used, as seen in Figure 4A (clipping) vs. Figure 4C (reflec-

tion). The impact of varying the available perturbation range on validation accuracy is shown in Figure 4B (clipping) and Figure 4D (reflection).

Very large perturbation ranges with clipping cause an increasing number of pixel locations to be clipped, and as expected, this leads to a progressive decrease in accuracy, but to a lesser extent than might be predicted based on visual inspection of the encoded results. As shown in Figure 4B, perturbation ranges up to -50% to +50% do not appreciable decrease validation accuracy, and even very large perturbation ranges above -150% to +150% that render the examples uninterpretable to human inspection still produce validation accuracies around 0.95.

Very large perturbation ranges with reflection are expected to retain more useful training information than clipping, as clipping is intrinsically a lossy manipulation, while reflection need not be. Consistent with this, increasing the perturbation range with reflection causes a smaller decrease in validation accuracy than that seen with clipping, reaching a plateau of validation accuracy around 0.965 (Figure 4D).

4.4 Serial application of padding, perturbation, and orthogonal transformation

The central security foundation of the present method involves sequential application of padding, perturbation, and orthogonal transformation to yield an aggregate non-orthogonal and nonlinear transformation. While the results presented in Figure 2 show that single transformations of padding, perturbation, or orthogonal transformation do not decrease validation accuracy, it is critical to test the impact of different forms of padding and perturbation performed in series with orthogonal transformation, in a typical sequence of (padding → perturbation → orthogonal transformation [using the fixed shuffling form of orthogonal transformation in the examples shown]).

4.4.1 Different padding approaches combined with fixed perturbation and orthogonal transformation

The different forms of padding described above (fixed padding, adjusted padding, and non-fixed padding) may be applied together with fixed perturbation and orthogonal transformation, so each of these approaches was tested to train models on data encoded with the sequence (padding → fixed perturbation → orthogonal transformation).

As shown in Figure 5A, application of fixed padding together with fixed perturbation and fixed shuffling produces a very small but significant decrease in validation accuracy (0.9754 for the encoded data compared with 0.9769 for control, a relative decrease in validation accuracy of 0.15%). Adjusted padding (see above for detailed description) using a potential adjustment of -20% to +20% together with fixed perturbation and fixed shuffling yielded a similar validation accuracy (0.975). Application of non-fixed padding (a different random pad for every training and test example) together with fixed perturbation and fixed shuffling yielded a slightly lower validation accuracy (0.9742 for the encoded data, a relative decrease in validation accuracy of 0.29%).

Similar results were obtained when random orthogonal matrix transformation was used instead of fixed shuffling in the above combinations of padding and fixed perturbation (data not shown).

4.4.2 Different padding approaches combined with non-fixed perturbation and orthogonal transformation

The different forms of padding (fixed padding, adjusted padding, and non-fixed padding) may also be applied together with non-fixed perturbation and orthogonal transformation, so each of these approaches was tested to train models on data encoded with the sequence (padding → non-fixed perturbation → orthogonal transformation [using the fixed shuffling form of orthogonal transformation in the examples shown]).

As shown in Figure 6A, application of fixed padding together with non-fixed perturbation and fixed shuffling produces a small but significant decrease in validation accuracy (0.9750, a relative decrease in validation accuracy of 0.23%). Adjusted padding using a potential adjustment of -20% to +20% together with non-fixed perturbation and fixed shuffling yielded a similar validation accuracy (0.9750, a relative decrease in validation accuracy of 0.21%). Application of non-fixed padding together with non-fixed perturbation and fixed shuffling

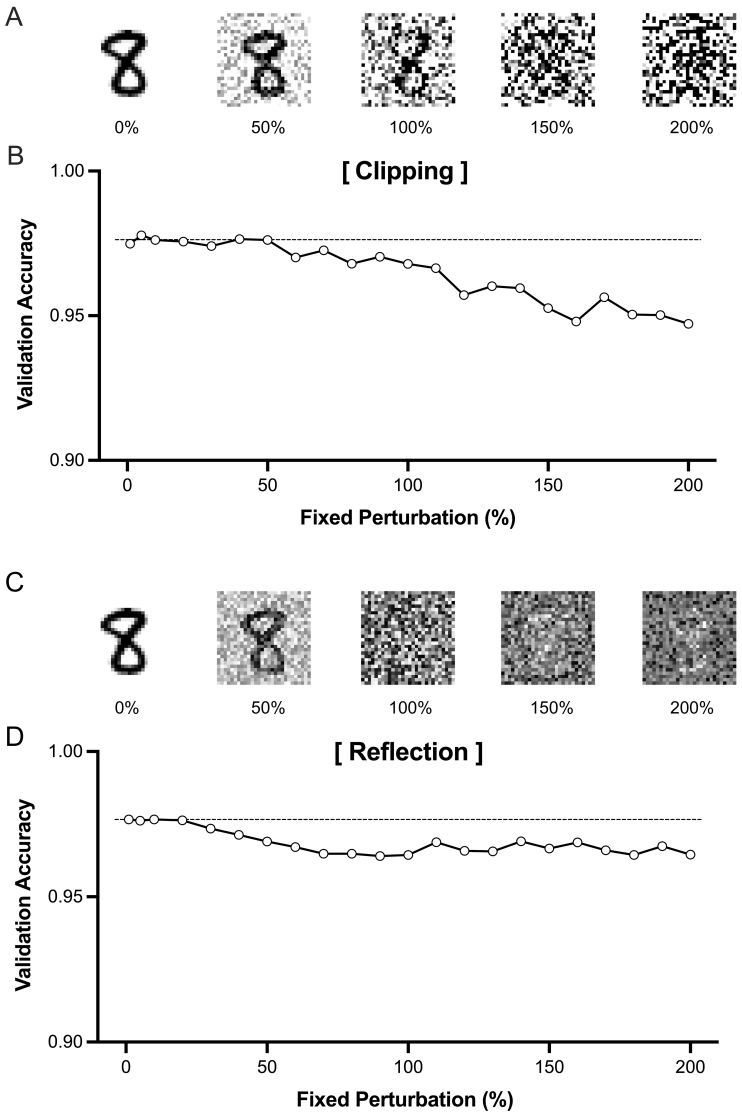

Figure 4: Variation of perturbation range.
A. Illustration of perturbation ranges from 0% (control) to 200% (-200% to +200%), processed with *clipping* to handle values beyond the pixel value range.
B. Validation accuracy of models trained with a single-step transformation using fixed perturbation with clipping, from 0% to 200%. Horizontal dotted line indicates the control baseline.
C. Illustration of perturbation ranges from 0% to 200%, processed with *reflection* to handle values beyond the pixel value range.
D. Validation accuracy of models trained with a single-step transformation using fixed perturbation with reflection, from 0% to 200%. Horizontal dotted line indicates the control baseline.

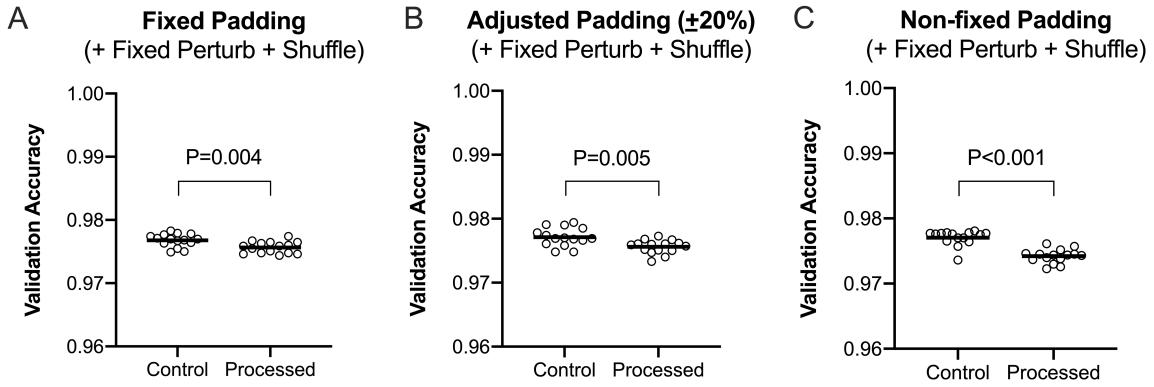

Figure 5: Different padding approaches combined with fixed perturbation and fixed shuffling.
A. *Fixed padding* (the same 2x28 pad appended to every example), followed by fixed perturbation (-5% to +5%), followed by fixed shuffling. Mean validation accuracy was 0.9754 ± 0.001 (Processed) compared with 0.9769 ± 0.0011 (Control) (P=0.004).
B. *Adjusted padding* (a 2x28 pad that is randomly perturbed within the bounds [-20% to +20%] prior to appending to each example), followed by fixed perturbation of the examples (-5% to +5%), followed by fixed shuffling. Mean validation accuracy was 0.9756 ± 0.0011 (Processed) compared with 0.9771 ± 0.0014 (Control) (P=0.005).
C. *Non-fixed padding* (a different 2x28 random pad appended to each example), followed by fixed perturbation (-5% to +5%), followed by fixed shuffling. Mean validation accuracy was 0.9742 ± 0.001 (Processed) compared with 0.9770 ± 0.0012 (Control) (P<0.001).

yielded a lower validation accuracy (0.9738 for the encoded data, a relative decrease in validation accuracy of 0.35%).

Similar results were obtained when random orthogonal matrix transformation was used instead of fixed shuffling in the above combinations of padding and non-fixed perturbation (data not shown).

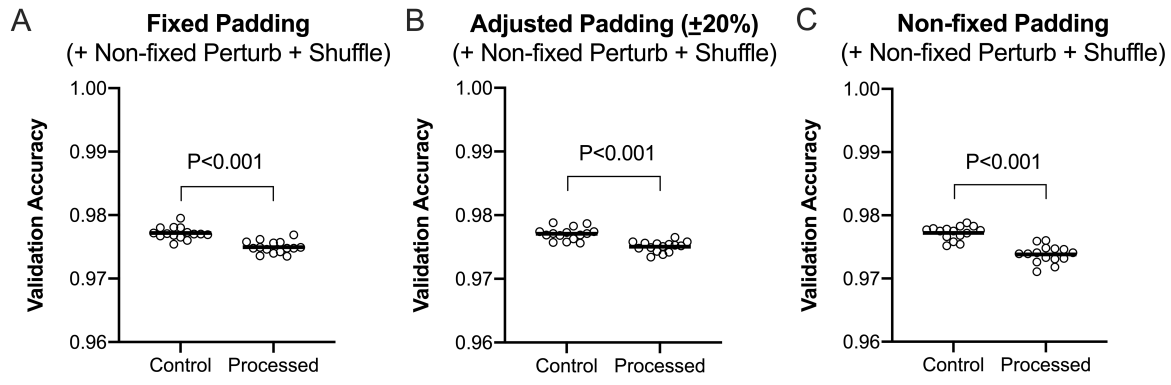

Figure 6: Different padding approaches combined with non-fixed perturbation and fixed shuffling.
A. *Fixed padding*, followed by *non-fixed perturbation* (-5% to +5%), followed by fixed shuffling. Mean validation accuracy was 0.9750 ± 0.001 (Processed) compared with 0.9772 ± 0.001 (Control) (P<0.001).
B. *Adjusted padding*, followed by *non-fixed perturbation* of the examples (-5% to +5%), followed by fixed shuffling. Mean validation accuracy was 0.9750 ± 0.0008 (Processed) compared with 0.9771 ± 0.0010 (Control) (P<0.001).
C. *Non-fixed padding*, followed by *non-fixed perturbation* (-5% to +5%), followed by fixed shuffling. Mean validation accuracy was 0.9738 ± 0.0014 (Processed) compared with 0.9772 ± 0.0011 (Control) (P<0.001).

4.5 Training time

Training times were tested to compare computation time for control unencoded MNIST with the same model specification trained using MNIST encoded with sequential padding, perturbation, and fixed shuffling.

Model training with MNIST encoded with fixed padding followed by fixed perturbation followed by fixed shuffling showed no significant difference in training times when compared with control MNIST (25.92 ± 0.57 sec for transformed MNIST vs. 26.06 ± 0.65 sec for control, Figure 8). The mean processing time required to perform encoding with these three transformations across the training and test tensors (representing a one-time processing time for a given project) was 22.49 ± 0.23 sec, which represents about 0.3 msec per MNIST example. Of note, the code used to perform these serial transformations was not optimized to minimize the required processing time for encoding, as the brief processing times required here were acceptable for the present work.

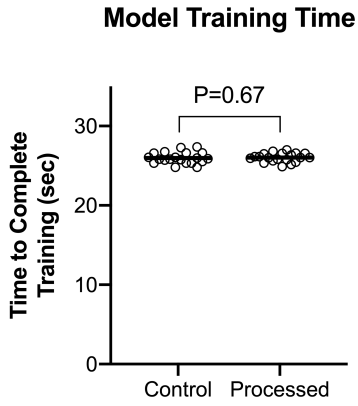

Figure 7: Model training time.
Model training for a fixed padded (2x28), fixed perturbed (-5% to +5%), and fixed shuffled data set (Processed) was 25.92 ± 0.57 sec, compared with 26.06 ± 0.65 sec for control (P=0.67). For this set of transformations (fixed padding, fixed perturbation, and fixed shuffling), processing of MNIST prior to model training took 22.49 ± 0.23 sec, which represents a one-time processing step prior to training for a given project. Processing times were tested on a Google Colab+ server running on a NVIDIA Tesla T4 GPU, with 20 separate model trainings per condition.

4.6 Encoded Dataset Example

An example dataset encoded with the above methods (beginning with a synthetic unknown dataset) is available at: https://drive.google.com/drive/folders/1Q0xspKzQCgP4ouYWcHGJcsLCa4o6pHjd

The encoded dataset is stored as separate .npy files for x_train, y_train, x_test, and y_test, and consists of a 60,000 train / 10,000 test split with (58 x 58) pixel grayscale examples and labels in 10 categories. There are 6,000 examples in each of the train categories, and 1,000 examples in each of the test categories. Using the fully-connected ANN architecture as above, direct training on this encoded dataset yielded a validation accuracy of 0.9725 ± 0.0029. Loading the .npy filetype into Python may be performed with *numpy.load* (https://numpy.org/doc/stable/reference/generated/numpy.load.html).

5 Discussion

5.1 Summary

A method for secure encoding of datasets for machine learning operations is described here in detail and shows very little reduction in ANN training performance when compared to training with unencoded datasets. The

individual transformations of the method applied separately do not significantly reduce validation accuracy. Training on datasets transformed by sequential application of padding followed by perturbation followed by orthogonal transformation results in validation accuracies that are only slightly lower than those seen with unmodified control datasets (with relative decreases in accuracy of 0.15% to 0.35%). Training time is similar between encoded and control datasets.

5.2 Security aspects of the method

The non-orthogonal and nonlinear orthogonal transformation in the described method creates an extraordinarily large search space in which a potential attacker would have to determine the encoding transformation. For example, index shuffling of a given array yields a single result at random out of a very large set of possible shuffled arrays, with the number of possible arrays represented by the factorial of the length of the array. Thus, even for small example arrays such as those in MNIST, with (28x28 = 784 pixel) examples, the total number of possible index shuffles for an example is 784!, which is 3.19 x 10^{1930} possible index shuffles, a number vastly larger than the number of guesses required to brute-force AES-256 (roughly 10^{76}). Beyond the specific case of index shuffling, the space of all possible random orthogonal matrix transformations for a given input vector size is far greater.

While orthogonal matrix transformation generates an enormous search space that cannot reasonably be solved by brute-force methods in the absence of knowledge of before / after examples, if an attacker is able to subject vectors to an unknown orthogonal matrix transformation and obtain the resulting encoded vectors, they will be able to trivially solve for the unknown transformation. For any linear transformation with an unknown square matrix with shape [n x n], which includes any possible orthogonal matrix transformation, an attacker with n input vectors each of length n and the corresponding n encoded vectors of length n will be able to solve for the unknown matrix using standard linear algebraic techniques (see below for details).

Combining one or more non-orthogonal and nonlinear transformations such as padding (fixed or non-fixed) or perturbation (fixed or non-fixed) with one or more orthogonal transformations yields an aggregate non-orthogonal and nonlinear transformation that is not susceptible to the above algebraic solution to determine the unknown transformation.

Different forms of padding transformation are available for use with perturbation and fixed shuffling, and these different types of padding provide different levels of security. For fixed padding, with the same pad applied to each example in a tensor, the search space is increased for any attempt to decode the relationship between the encoded dataset and the original, and the change in vector length from input vectors to encode vectors prevents an attacker solving for a square linear matrix transformation. However, if the original dataset does not have any pixel values that are invariant across all examples in the tensor, a methodical search pixel by pixel for invariant pixel values across a full tensor would allow these pixels to be ignored. It should be noted that for larger examples and datasets, this analysis is not computationally trivial. However, for datasets such as MNIST, there are many pixel locations that have values that are invariant across all examples (e.g., in the case of MNIST, the white pixels in the corners of the examples).

Non-fixed padding or adjusted padding may be applied to achieve higher levels of security, at a very small cost in terms of validation accuracy, as shown here. Non-fixed and adjusted padding serve to increase the search space by increasing the number of shuffled pixels per example in a fashion that is not susceptible to reduction of this search space by ignoring invariant pixels. The added extrinsic noise created by non-fixed padding or adjusted padding extends across the example dimension of the tensors and is thus different for each element in the training, validation, and inference examples.

Perturbation also adds extrinsic noise that serves to foil attempts to reduce the search space. Fixed perturbation does not suffer from an 'ignore invariant pixel values' attack as discussed above for fixed padding, because a fixed perturbation is an interaction between a fixed array of perturbation functions with the pixels across the examples, which are themselves variable. The only situation where fixed perturbation would lead to identical pixels across an entire tensor would be in situations where pixels in the source data are themselves invariant (like the white corners of MNIST examples). Regardless, non-fixed perturbation adds an even greater degree of extrinsic noise across the tensor by varying perturbation from example to example,

further reducing the likelihood that an attacker could reduce the search space, at little cost in terms of validation accuracy, as shown in Figures 5 and 6.

Independent security testing may be performed on a publicly accessible encoded dataset (see section 4.6 above).

5.3 Adaptation to convolutional neural networks

The current paper explores the performance of various aspects of the padding, perturbation, and orthogonal transformation method in the context of fully-connected ANNs. It is also possible to adapt this approach to encoding for use with convolutional neural networks (CNNs) (Anonymized, 2022). The operations of convolutions and pooling in CNNs use the spatial relationships of data elements, so these operations cannot be performed after an orthogonal transformation such as fixed shuffling in a fashion that is agnostic to the fixed shuffling algorithm. Instead, convolution and pooling operations can be performed prior to application of padding, perturbation, and orthogonal transformation, with fixed shuffled tensors created to store multiple results of convolution and pooling (Anonymized, 2022). Alternatively, the algorithm used to apply padding, perturbation, and an orthogonal transformation such as fixed shuffling may be encrypted by conventional methods, transmitted securely, then decrypted by authorized users to apply convolution and pooling operations, using the decrypted algorithm as a lookup table that enables these operations on encoded data (Anonymized, 2022).

5.4 Alternative approaches

Various alternative approaches to the problem of securing data for model training have been proposed, with recent work focused on the areas of *federated learning*, the use of data treated with *fully homomorphic encryption*, and application of *orthogonal transformation*.

5.4.1 Federated learning

Federated learning is a well-established approach for performing distributed model updates using data on local devices, wherein training data do not have to be transmitted from the local device to a central server. In the federated learning framework, a model and weights are transmitted to the local device, weights are updated by additional local training, and then the updated models and weights returned to the central server (Li et al., 2020; Rodríguez-Barroso et al., 2020). While this approach has the advantage of not needing to transmit data from a local device to a central server, and has found wide application in updating models using rich data on more computing-constrained systems such as mobile devices, it has certain disadvantages when it comes to the problem of model training of private data. Software provided by the central server must be run to train or update models on local systems that maintain the private data, and this software has direct access to the private data. This means that the party maintaining the private data ('Party A') must trust the party providing the software ('Party B'). Party A must trust Party B to run software behind the Party A's firewall and additionally trust this software to have direct access to Party A's private data. Party A additionally must trust Party B not to transmit any aspect of Party A's private data back to Party B's servers along with the updated models and weights that have to be transmitted from behind Party A's firewall to Party B's server. Finally, for projects in which new models need to be trained from scratch, the distributed approach of federated learning does not solve the basic issue that training neural network models on large datasets typically requires data transmission to cloud resources.

In addition to the security issues discussed above, federated learning can have an important adverse business ramification: because federated learning is particularly well-suited for updating models on a wide range of data sources, this means that the business value of private data provided by any one organization will tend to be devalued.

5.4.2 Fully homomorphic encryption

Fully homomorphic encryption (FHE) represents a family of approaches that allow mathematical manipulation of encrypted data such that when a manipulated ciphertext is decrypted, the result is identical to

an analogous mathematical manipulation performed on the original data prior to encryption (Lauter, 2021; Munjal & Bhatia, 2022; Alloghani et al., 2019). There have been initial explorations of using FHE in the context of secure machine learning (Bost et al., 2014; Graepel et al., 2013; Lauter, 2021; Nandakumar et al., 2019; Papernot et al., 2016), but FHE requires profound alterations to machine learning algorithms that yield compute times that are many orders of magnitude higher than required for work on unencrypted data using conventional methods (Nandakumar et al., 2019). For example, training on MNIST in an initial exploration of using FHE with a neural network required downsampling of the MNIST dataset to 8x8 pixel examples and use of a cyclotomic ring encryption considered to not provide sufficient security ($\phi(m) = 600$), and despite these simplifications, compute times for the neural network ranged from 40 minutes to 1.5 days (Nandakumar et al., 2019). When a more secure FHE algorithm was tested ($\phi(m) = 27,000$), the time to process a single neuron in the first layer was 15 minutes (Nandakumar et al., 2019). In another experiment to perform unsupervised learning with K-means-clustering on an FHE-encrypted dataset, single-thread runtimes were estimated to be on the timescale of months (Jäschke & Armknecht, 2019).

5.4.3 Orthogonal transformation

Orthogonal transformation has been proposed as a one-step encoding for machine learning applications, specifically by performing QR factorization on a digital image file of appropriate size and using the resulting orthogonal matrix to transform input vectors into encoded vectors (Chen, 2019). Using an orthogonal matrix to transform input vectors into encoded vectors maintains the angles between vectors and the lengths of vectors, by transforming the relative position of the vectors in the coordinate system (representing a rotation and/or reflection of the coordinate system, or in the specific case of index shuffling, shuffling of the coordinate axes). Because the relationships between vector angles and vector lengths are preserved in this transformation, machine learning accuracy is expected to be unaffected, and we show here that this is indeed the case in practice, for both the specific case of index shuffling and the broader case of random orthogonal matrix transformation.

However, as described above, an orthogonal matrix transformation in isolation suffers from an important security liability. Orthogonal matrix transformations are a subset of square matrix linear transformations, and any unknown square matrix linear transformation T of shape [n x n] may be explicitly determined in the presence of n vectors of length n before transformation and n corresponding vectors of length n after transformation, using standard linear algebraic manipulation. Briefly, the n input vectors of length n are packaged as column vectors into an [n x n] input matrix *A_input*, and the encoded vectors are packaged into a corresponding [n x n] encoded matrix *A_enc*. We therefore have the relationship $A\_enc = T \cdot A\_input$, and dotting both sides of the equation by the inverse of the input matrix, *A_input_inv*, yields the solution for the unknown matrix, $T = A\_enc \cdot A\_input\_inv$. Thus, an attacker that is able to present n examples to an algorithm consisting solely of an orthogonal matrix transformation (or any other linear square matrix transformation) and obtain n transformed examples will be able to algebraically solve for the transformation, breaking the security of this method of encoding.

While the present method utilizes orthogonal transformation, it does so together with the non-orthogonal and nonlinear transformations of padding and perturbation, yielding an aggregate non-orthogonal and nonlinear transformation that is not subject to the above method of attack.

5.4.4 Other approaches

Other variations on the theme of homomorphic encryption with additional constraints have been tested with neural networks with improved performance results, such as the approach of *leveled homomorphic encryption* (Bos et al., 2013), together with tailoring of network structures to the specific encryption approaches used in order to make the required computations more tractable for neural network training (Gilad-Bachrach et al., 2016).

Another proposed approach adapts multi-party computation techniques (Goldreich, 2004), in which encrypted data is passed back and forth between parties, updating calculations layer by layer in stepwise fashion, with the sending party required at each step to decrypt the data, apply a nonlinear transformation, encrypt the result and return this to the other party (Barni et al., 2006; Orlandi et al., 2007). While recent

adaptations of this technique have improved the computational efficiency, the approach still requires bidirectional transmission of very large amounts of data, substantially longer training times when compared with plaintext training, and technical sophistication on both ends of the data exchange to implement the method (Keller & Sun, 2021).

5.5 Advantages of the presented methods

Compared to alternative approaches, the method presented here has certain advantages. The secure encoding process described allows for use of unchanged machine learning systems, without fundamental alteration of existing frameworks. The method also allows for customization of the encoding process so that the trade-off between security and very small decrements in accuracy may be adjusted according to a given project's needs. There is no loss in training time when using the encoding described here, and only minimal one-time computations are needed to perform the encoding steps.

Importantly, the specific encoding steps used must be applied in similar fashion to the training and test tensors and any inference examples. This requirement to encode inference data protects the business value of the encoded training data used in a collaboration, because the collaboration may be set up such that the party that contributes encoded data maintains the secret of the encoding process, which must be securely accessed in order to perform inference.

5.6 Conclusion

A technique for securely encoding examples with non-orthogonal and nonlinear transformation is presented that enables direct artificial neural network (ANN) training on encoded datasets. No modifications to standard neural network architecture are needed. Individual, single-step transformations of the method do not significantly reduce validation accuracy, and training on datasets transformed by sequential padding, perturbation, and random orthogonal transformation yields only very slightly reduced validation accuracies. No differences in training times are seen between encoded and control datasets. The described method is expected to have applications in machine learning wherever data security is a requirement.

References

Mohamed Alloghani, Mohammed M. Alani, Dhiya Al-Jumeily, Thar Baker, Jamila Mustafina, Abir Hussain, and Ahmed J. Aljaaf. A systematic review on the status and progress of homomorphic encryption technologies. *Journal of Information Security and Applications*, 48:102362, October 2019. ISSN 2214-2126. doi: 10.1016/j.jisa.2019.102362.

Authors Anonymized. Nonprovisional patent application on file with USPTO. February 2022.

Boris Babenko, Akinori Mitani, Ilana Traynis, Naho Kitade, Preeti Singh, April Y. Maa, Jorge Cuadros, Greg S. Corrado, Lily Peng, Dale R. Webster, Avinash Varadarajan, Naama Hammel, and Yun Liu. Detection of signs of disease in external photographs of the eyes via deep learning. *Nature Biomedical Engineering*, pp. 1–14, March 2022. ISSN 2157-846X. doi: 10.1038/s41551-022-00867-5.

M. Barni, C. Orlandi, and A. Piva. A privacy-preserving protocol for neural-network-based computation. In *Proceedings of the 8th Workshop on Multimedia and Security*, MM&Sec '06, pp. 146–151, New York, NY, USA, September 2006. Association for Computing Machinery. ISBN 978-1-59593-493-2. doi: 10.1145/1161366.1161393.

Joppe W. Bos, Kristin Lauter, Jake Loftus, and Michael Naehrig. Improved Security for a Ring-Based Fully Homomorphic Encryption Scheme. In Martijn Stam (ed.), *Cryptography and Coding*, Lecture Notes in Computer Science, pp. 45–64, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-45239-0. doi: 10.1007/978-3-642-45239-0_4.

Raphael Bost, Raluca Ada Popa, Stephen Tu, and Shafi Goldwasser. Machine Learning Classification over Encrypted Data. Technical Report 331, 2014.

Monchu Chen. Methods and processes of encrypted deep learning services, March 2019.

Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research. *IEEE Signal Processing Magazine*, 29(6):141–142, November 2012. ISSN 1558-0792. doi: 10.1109/MSP.2012.2211477.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 201–210. PMLR, June 2016.

Oded Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, Cambridge, UK ; New York, 1st edition edition, May 2004. ISBN 978-0-521-83084-3.

Thore Graepel, Kristin Lauter, and Michael Naehrig. ML Confidential: Machine Learning on Encrypted Data. In Taekyoung Kwon, Mun-Kyu Lee, and Daesung Kwon (eds.), *Information Security and Cryptology – ICISC 2012*, pp. 1–21, Berlin, Heidelberg, 2013. Springer. ISBN 978-3-642-37682-5. doi: 10.1007/978-3-642-37682-5_1.

Varun Gulshan, Lily Peng, Marc Coram, Martin C. Stumpe, Derek Wu, Arunachalam Narayanaswamy, Subhashini Venugopalan, Kasumi Widner, Tom Madams, Jorge Cuadros, Ramasamy Kim, Rajiv Raman, Philip C. Nelson, Jessica L. Mega, and Dale R. Webster. Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs. *JAMA*, 316(22):2402–2410, December 2016. ISSN 1538-3598. doi: 10.1001/jama.2016.17216.

Angela Jäschke and Frederik Armknecht. Unsupervised Machine Learning on Encrypted Data. In Carlos Cid and Michael J. Jacobson Jr. (eds.), *Selected Areas in Cryptography – SAC 2018*, pp. 453–478, Cham, 2019. Springer International Publishing. ISBN 978-3-030-10970-7. doi: 10.1007/978-3-030-10970-7_21.

Marcel Keller and Ke Sun. Secure Quantized Training for Deep Learning, July 2021. Comment: 17 pages.

Patty Kostkova, Helen Brewer, Simon de Lusignan, Edward Fottrell, Ben Goldacre, Graham Hart, Phil Koczan, Peter Knight, Corinne Marsolier, Rachel A. McKendry, Emma Ross, Angela Sasse, Ralph Sullivan, Sarah Chaytor, Olivia Stevenson, Raquel Velho, and John Tooke. Who Owns the Data? Open Data for Healthcare. *Frontiers in Public Health*, 4, 2016. ISSN 2296-2565.

Kristin E. Lauter. Private AI: Machine Learning on Encrypted Data. Technical Report 324, 2021.

Li Li, Yuxi Fan, Mike Tse, and Kuo-Yi Lin. A review of applications in federated learning. *Computers & Industrial Engineering*, 149:106854, November 2020. ISSN 0360-8352. doi: 10.1016/j.cie.2020.106854.

Kundan Munjal and Rekha Bhatia. A systematic review of homomorphic encryption and its contributions in healthcare industry. *Complex & Intelligent Systems*, May 2022. ISSN 2198-6053. doi: 10.1007/s40747-022-00756-z.

Karthik Nandakumar, Nalini Ratha, Sharath Pankanti, and Shai Halevi. Towards Deep Neural Network Training on Encrypted Data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 0–0, 2019.

Claudio Orlandi, Alessandro Piva, and Mauro Barni. Oblivious Neural Network Computing via Homomorphic Encryption. *EURASIP Journal on Information Security*, 2007(1):1–11, December 2007. ISSN 1687-417X. doi: 10.1155/2007/37343.

Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the Science of Security and Privacy in Machine Learning. *arXiv:1611.03814 [cs]*, November 2016.

Mark Phillips. International data-sharing norms: From the OECD to the General Data Protection Regulation (GDPR). *Human Genetics*, 137(8):575–582, August 2018. ISSN 1432-1203. doi: 10.1007/s00439-018-1919-7.

Ryan Poplin, Avinash V. Varadarajan, Katy Blumer, Yun Liu, Michael V. McConnell, Greg S. Corrado, Lily Peng, and Dale R. Webster. Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nature Biomedical Engineering*, 2(3):158–164, March 2018. ISSN 2157-846X. doi: 10.1038/s41551-018-0195-0.

Nuria Rodríguez-Barroso, Goran Stipcich, Daniel Jiménez-López, José Antonio Ruiz-Millán, Eugenio Martínez-Cámara, Gerardo González-Seco, M. Victoria Luzón, Miguel Angel Veganzones, and Francisco Herrera. Federated Learning and Differential Privacy: Software tools analysis, the Sherpa.ai FL framework and methodological guidelines for preserving data privacy. *Information Fusion*, 64:270–292, December 2020. ISSN 1566-2535. doi: 10.1016/j.inffus.2020.07.009.

Kim Theodos and Scott Sittig. Health Information Privacy Laws in the Digital Age: HIPAA Doesn't Apply. *Perspectives in Health Information Management*, 18(Winter):1l, December 2020. ISSN 1559-4122.