MUTE: Inter-class Ambiguity Driven Multi-hot Target Encoding for Deep Neural Network Design

Mayoore S. Jaiswal
IBM
Austin, TX
mayoore.s.jaiswal@ibm.com

Bumsoo Kang
KAIST
Daejeon, South Korea
bumsoo@nclab.kaist.ac.kr

Jinho Lee
Yonsei University
Seoul, South Korea
leejinho@yonsei.ac.kr

Minsik Cho
IBM
Austin, TX
minsikcho@us.ibm.com

Abstract

Target encoding is an effective technique to boost performance of classical and deep neural networks based classification models. However, the existing target encoding approaches require significant increase in the learning capacity, thus demand higher computation power and more training data. In this paper, we present a novel and efficient target encoding method, Inter-class Ambiguity Driven Multi-hot Target Encoding (MUTE), to improve both generalizability and robustness of a classification model by understanding the inter-class characteristics of a target dataset. By evaluating ambiguity between the target classes in a dataset, MUTE strategically optimizes the Hamming distances among target encoding. Such optimized target encoding offers higher classification strength for neural network models with negligible computation overhead and without increasing the model size. When MUTE is applied to the popular image classification networks and datasets, our experimental results show that MUTE offers better generalization and defense against the noises and adversarial attacks over the existing solutions.

1. Introduction

Artificial intelligent systems require efficient deep neural network design methodologies that can learn semantics of the training dataset, generalize well, and resist adversarial attacks. However, existing methods have been shown to learn dataset bias [30, 29, 24], and failed to deliver sufficient generalization capability. Poor generalization makes models unpredictable, causes potential ethical issues, and mis-guides neural network design [36, 10]. To tackle the generalization problems, target encoding has been studied for both conventional machine learning and deep neural network architectures and proven to be highly effective [2, 9, 15, 4]. Yet, many prior works in target encoding require a long encoding sequence (which increases the model size) and fail to tailor the encoding for a given task or dataset. Furthermore, they do not investigate the effects of different target encodings against noisy data and adversarial attacks.

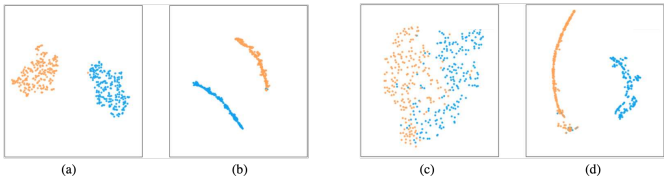

Figure 1. TSNE [25] visualizations of a ConvNet (refer Section 4.2 for architecture) with one-hot encoding trained on MNIST (a, c) and a ConvNet with MUTE trained on MNIST (b, d). Classes that are well-separated in features space in models trained with one-hot encodings (a), remain well separated in models trained with MUTE (b). However, classes not-well-separated in features space in models trained with one-hot encodings (c), get well-separated in feature space when trained with MUTE (d).

In this work, we propose MUTE, a systematic approach to make deep learning models generalize better by optimizing the target encoding [2, 9, 15, 4]. Unlike the conventional one-hot method where the Hamming distance between labels is fixed at 2, MUTE generates a multi-hot encoding by exploiting the expression power of a given output encoding length. MUTE and one-hot encoding have the same number of output encoding length. So, models trained from both methods have same size, however MUTE effectively utilizes the available learning capacity.

MUTE strategically extracts the *ambiguity* between pairs of classes in a dataset, and leverages on that information to

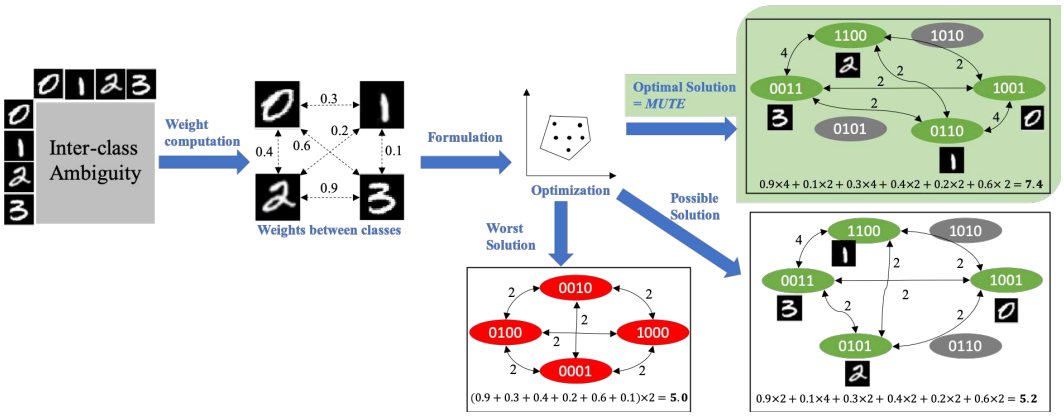

Figure 2. MUTE generation. The inter-class ambiguity for a given data is used to compute weights between classes. These weights are used in Algorithm 1 to optimally assign target codes to classes. The chosen set of MUTE are highlighted in green color. Details are in Section 3.

obtain a multi-hot encoding such that semantically closer classes are forced to be further apart in the label space in terms of the Hamming distance. MUTE ensures that Stochastic Gradient Decent (SGD) algorithm extracts distinctive features between two easy-to-confuse classes, which in turn reduces the chance of mis-prediction under noisy and noiseless conditions. Learning good feature representations also help to utilize the features for any downstream tasks [35].

An intuitive explanation is that MUTE makes the Hamming distance between labels larger (and controllable) than one-hot encoding. With one-hot encoding, a large error in a single logit could alter the classification result. Whereas in MUTE, a large error in a single logit alone would not alter the result, since there are still $k-1$ more logits that has to agree on the decision (like error correcting codes). This, we conjecture, enables MUTE to learn separable features.

Figure 1 illustrates the high-level idea in MUTE by showing a case where two classes with higher-level of misclassification are assigned to the multi-hot encodings with a larger Hamming distance. In order to find such high-quality encoding in MUTE, we formulate the encoding generation as a subgraph isomorphism problem [32] and develop efficient heuristics to solve the combinatorics. Figure 2 gives an overview of the MUTE generation which is discussed in detail in Section 3.

We evaluate MUTE by training multiple convolutional neural network (CNN) architectures with benchmark datasets such as MNIST [23], CIFAR-10 [19] and ICON-50 [13], and testing on a validation set which consists of original, noisy (i.e., negative of the original images, Gaussian blurred images, images with salt-and-pepper noise) and adversarial images similar to real-world conditions. Our results show that the models built by MUTE have comparable accuracy on original clean images, yet delivered better test accuracy against noisy images than the traditional one-hot encoding and prior work, especially when the learning capacity of a model is limited. Our contributions in MUTE include the following: **a)** novel target encoding scheme based on inter-class ambiguity in a given dataset and weighted Hamming distance, **b)** effective heuristics for subgraph isomorphism in the target encoding context, **c)** study the impact of MUTE on various datasets and CNN architectures, and **d)** comprehensive study on the effects of our target encoding scheme on both noisy and adversarial images.

2. Related Work

In this section, we review various encoding techniques used in classification problems.

Multi-class Single-label: Each sample in a dataset belongs to only one class. One-hot (1-of-N) encoding is commonly used in this case. One-hot encoding represents the target labels numerically without any semantic meaning and is prone to noisy inputs or adversarial attacks [13, 28, 26, 11]. MUTE is a multi-bit code that takes dataset semantics into account.

Multi-class Multi-label: This is the superposition of multiple one-hot encodings for the case where a sample belongs to multiple semantic classes [15, 4]. This contrasts with multi-hot encoding (i.e. MUTE) where a sample belongs to only one class, but represented in multiple bits.

Label Embedding: This is a technique to embed target labels with meta information like attributes [2] or hierarchies [31] to capture various structures of the output space at the cost of additional labelling and expert knowledge [9, 15]. In some cases, the embedding is jointly learned from input and output data [3, 33]. MUTE does not need any additional meta-data but leverages on the dataset semantics for its performance boost.

Target Encoding: These are methods that explore alternatives to one-hot encoding [18, 7, 22, 21, 5, 27], similar to MUTE. Yet, the key innovations in MUTE over them are as follows: **a)** MUTE has the same encoding length as one-hot unlike [18], and could be used as a drop-in-replacement of

any existing one-hot encoding, **b)** MUTE strategically optimizes the encoding w.r.t. the given inter-class characteristics.

Target Encoding in the context of Error-Correcting Output Codes (ECOC) has been extensively studied in [7, 22, 21, 5, 27] where ECOC concept is applied to represent output labels. While these approaches benefit from the error-correcting capability (as in MUTE), they are not optimized for a given task or dataset. Unlike MUTE, they do not capture the inherent characteristics of the dataset, specifically the relationships between classes, thus not necessarily increasing the predictive power of a network model. For example, [35] used Hadamard codes to learn deep representations and showed that such presentations are more separable than the ones learned with one-hot encodings, but failed to extract/utilize the useful information from the dataset.

Also, there is recent interest in exploring alternatives to one-hot encoding for multi-class classification problems with a focus on improving the performance against adversarial examples. In the multi-way encoding method [18], the authors proposed to use datatset-oblivious Random Orthogonal (RO) encodings [18]. RO encodings are in real-numbers and have larger encoding dimension than the number of output classes (which increases the number of trainable weights), whereas MUTE has the same encoding dimension as the number of output classes. Unlike [18], MUTE does not require additional adversarial training.

3. Inter-class Ambiguity Driven Multi-hot Target Encoding

We propose a new target encoding system, MUTE, where multiple output bits are activated. For an $N-$class classification problem with MUTE, the output layer has $N-$bits, out of which K bits are 1s (hereafter referred to as $K-$hot), where $K > 1$. The specific target codes are generated using Algorithm 1. Each output bit has a bounded non-linear activation function and trained such that the binary cross entropy loss between the prediction and target code is minimized.

When a deep neural network is trained with back-propagation, it is being optimized to learn features that distinguish classes. When it is able to learn distinct features, these classes are well separated in feature space and have a low probability of mis-prediction (i.e., Figure 1(a), (b) and (d)). However, when a network is not able to learn distinguishable features, these classes are not well separated, which may result in low accuracy (i.e., Figure 1(c)). The **core idea** of MUTE is to identify such ambiguous classes and assign target codes that have large Hamming distances between them. By training with such target codes, the network is optimized to learn features to separate ambiguous classes. Having clear decision boundaries between classes reduces classification error and improves performance against noisy and adversarial inputs.

The neural network training using MUTE starts by computing the ambiguity between classes in a given dataset, which is then quantified into weights. After that, a set of target codes are generated such that the Hamming distances among all classes are maximized, and a pair of ambiguous classes (i.e., classes with higher weights between them) are assigned codes with larger Hamming distance. This increases the distance between classes in the feature space as illustrated in Figure 1, thereby reducing the probability of mis-prediction, enhancing the generalization of the classifier, and encouraging a model to learn the semantics rather than spatial correlation among pixels. Finally, any chosen deep neural network could be trained and tested with MUTE without increasing the number of parameters. MUTE generation is illustrated in Figure 2, which will be discussed in the following sections.

3.1. Generating Weights For MUTE

The objective of this step is to quantitatively identify ambiguous classes and assign weights by the level of ambiguity. Note that this step is optional, and can be skipped by setting $W_{ij} = 1$ in Equation 1. However, using the weights generated by this step gives better results as shown in Section 4.2.

A confusion matrix (CM) is a method to represent inter-class characteristics. CM is essentially a $N \times N$ matrix for a $N-$class dataset with the elements being the confusion metric between a pair of classes as shown in Figure 2 (top-left). CM can be obtained by inferencing using a validation set on an already trained target model. We use the method proposed in [17] to generate CM where the confusion-level between classes of a dataset can be determined by reconstructing data for each class using an autoencoder trained using class C_i and determining the reconstruction error for every other class C_j in the dataset.

Once CM is obtained for a given dataset, MUTE can convert it into weights to be used in Section 3.2. [17] provides the reconstruction error, r_{ij}, between two classes C_i and C_j in CM, which is small for ambiguous classes and large for unambiguous classes. The inter-class ambiguity between classes C_i and C_j can be inferred from $CM(i, j)$ and $CM(j, i)$. First, the diagonal of CM (self-error values) are subtracted. Since, CM is not symmetric, but we need only one ambiguity metric between classes C_i and C_j, we choose the minimum error between the upper and lower triangles of CM to consolidate and to give best-case scenario for ambiguity. In order for the Fast-convergence algorithm (discussed in Section 3.2) to focus separating on ambiguous classes, we remove r_{ij} from unambiguous pairs of classes by eliminating r_{ij} greater than threshold τ where $\tau = mean(r_{ij}) + \alpha \cdot std(r_{ij})$. The remaining r_{ij} are inversely scaled to obtain weights W_{ij} where $W_{ij} = \beta / r_{ij}$. The outcome of this process is illustrated on the top-left of Figure 2.

3.2. Generating MUTE

The next step in MUTE is to generate and assign a code to each class. Our goal is twofold: **a)** to generate a code-set[1] that maximizes the Hamming distances among all pairs of codes, and **b)** to assign a pair of codes with a larger Hamming distance to more ambiguous classes. In this light, we put the total minimum Hamming distance and the Hamming distances between codes with weights (obtained from Section 3.1) together for our objective function, Equation 1, to be maximized. Since weights indicate inter-class ambiguity, maximizing the objective function assigns a larger Hamming distance to a pair of more ambiguous classes.

$$W_{min}H_{min} + \sum_{i=0}^{N-1}\sum_{j=i+1}^{N} W_{ij}H_{ij} \quad (1)$$

where W_{min} denotes the weight of the minimum Hamming distance H_{min}, N denotes the total number of classes, W_{ij} and H_{ij} denote inter-class ambiguity and the Hamming distance between class i and class j, respectively. N and W_{ij} are given constants (i.e., W_{ij} is computed as in Section 3.1). W_{min} should be larger than W_{ij} because maximizing H_{min} drives the overall model performance. We set W_{min} as the number of pairs of classes, $N(N-1)/2$, in our experiments. In one-hot encoding, H_{min} is fixed at 2, while H_{min} is maximized in MUTE.

Figure 2 shows an example of possible outcomes from the optimization step. With one-hot encoding (bottom-center in red), all possible solutions have the same objective value, which is the lowest among the solutions illustrated. Figure 2 also shows the difference between the optimal and a possible solution in terms of the Hamming distance and the generated code. The optimal solution picks and assigns the codes to the four classes such that more ambiguous classes (i.e., 2 vs. 3 with weight 0.9) are assigned a pair of codes with a larger Hamming distance.

Optimization in Figure 2 can be formulated as an Integer Linear Programming (ILP) due to the discrete nature of the encoding itself, and solved by a commercial package. However, it requires a large amount of computation time to find an optimal solution because it explores the solution space in an exhaustive branch-and-bound manner. Yet, one useful observation on this combinatorics is that the distribution of Hamming distance values are very discrete and narrow, and has high-density only on a few integer values. Therefore, we expect that there should be a number of possible solutions that have the same objective values. In other words, this maximization problem has a wide and sparse solution space, allowing us to develop an efficient heuristic, *Narrow-and-converge* approach on top of *Fast-convergence* algorithm (Algorithm 1), which is inspired from [8] and finds a solution to Equation 1 significantly faster than ILP optimizations.

Narrow-and-converge approach narrows down the solution space and finds a solution with Fast-convergence algorithm. First, Equation 1 without weights (i.e., $W_{ij} = 1$) is optimized with ILP for a short time until it narrows down the solution space. The time limit is a user parameter that depends on the given number of hot bits(K) and total bits used to represent the classes in the output layer (N) (number of total bits and classes are same in our case). In our setting, it was sufficient to run ILP for 10 seconds on $4-$hot codes for $10-$classes, and 3 minutes for $20-$hot codes for $50-$classes. Second, we capture the intermediate code-set from the first step and shuffle them to obtain a number of objective values of Equation 1 with weights. We pick the code-set that maximizes the objective value to initialize Algorithm 1.

The Fast-convergence algorithm (Algorithm 1) visits all the codes in the code-set in a random permutation. Each time a code is visited, a candidate pool is generated by swapping a randomly selected 0 and 1 until the number of candidates becomes $NumCand$. The number of bits to swap is a user parameter, which defines a distance between the code and its candidate. Having a longer distance enables Algorithm 1 to explore a wider solution space. Since our goal was to quickly converge to one of the optimal solutions, we minimize the distance between the code and its candidate by swapping only one bit. Therefore, the maximum number of candidates becomes *the number of zeros* $\times$ *the number of ones*.

The algorithm picks the candidate that maximizes Equation 1 over the candidate pool, and replaces the code with the chosen candidate even if the chosen candidate is worse than the current one. This procedure is repeated until every code is visited. Then, it starts over from the best code-set observed from the whole cycle. The iteration is stopped when the objective function does not improve for $NumIter$ cycles. As a result of starting over from the best code-set of the previous cycle, and the algorithm keeps finding a better solution. The main difference between ILP optimization and Narrow-and-converge approach is the search strategy. ILP takes a breadth-first search approach to find an optimal solution, whereas our approach takes a depth-first search strategy. Figure 3 describes how this approach maximizes objective value over iterations in two cases, including $8-$hot codes for $20-$classes and $10-$hot codes for $50-$classes.

3.3. Training Method

MUTE could be used with any CNN model with no changes to the architecture. The only change required is to replace the softmax layer with a sigmoid layer. The CNN is trained by back-propagating the binary cross entropy loss at each bit. CNN is optimized by SGD with momentum. An overview of training a CNN with MUTE is given in

[1]Here we use the term *code* to denote the bits assigned for a single class, and the term *code-set* to indicate the collection of codes that are used to represent all the classes in the dataset.

Algorithm 1 Fast-convergence Algorithm

Input: $codeset_{init}$
Output: $codeset_{target}$
Parameters: $NumIteration, NumCandidates$

procedure FASTCONVERGENCE($codeset_{init}$)
  $codeset_{target} \leftarrow codeset_{init}$
  while not $StopCondition$ **do** ▷ StopCondition: no improvements for $NumIteration$
    $codeset_{new} \leftarrow codeset_{target}$
    while not every code in $codeset_{new}$ is updated **do**
      $code_{target} \leftarrow pick(codeset_{new})$ ▷ randomly picks a $code_{target}$ from $codeset_{new}$
      $candidates \leftarrow generate(code_{target})$ ▷ generate $NumCandidates$ of candidates
      $code_{candidate} \leftarrow find(candidates)$ ▷ find the best candidate
      $replace(code_{target}, code_{candidate})$ ▷ replace the $code_{target}$ with the best candidate
      if $hamm\_dist(codeset_{new}) > hamm\_dist_{best}$ **then**
        $hamm\_dist_{best} \leftarrow hamm\_dist(codeset_{new})$ ▷ the best objective value
    $codeset_{target} \leftarrow \underset{codeset}{\text{argmax}}(hamm\_dist_{best})$ ▷ update the target code-set
  return $codeset_{target}$ ▷ the optimized target code-set

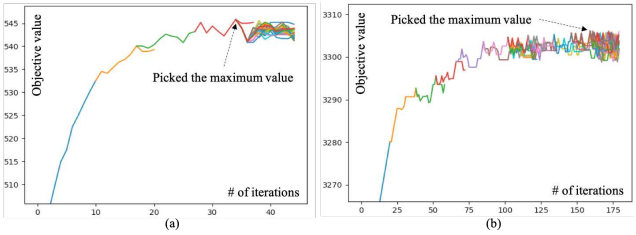

Figure 3. Convergence trend of Fast-convergence algorithm. Examples of (a) $8-$hot for $20-$classes, and (b) $10-$hot for $50-$classes.

Figure 4(a). In this method, the number of neurons in the CNN and the computational complexity is the same as using one-hot target encoding.

3.4. Inferencing Method

As shown in Figure 4(b), an input image is forward propagated through the trained model. The output of the sigmoid layer is thresholded by setting all but the $top - K$ activated bits to 0, where K is the number of hot bits used in the MUTE. Then, the Euclidean distance between the thresholded output and the code-set is computed. The label corresponding to the closest MUTE code is the classification result. We experimented with binarizing the $top - K$ output, computing $L1$ distance between output code and the code-set, and setting the label to be the closest code's label (any ties were broken at random). However, empirically we found that keeping the $top - K$ values, setting the rest to 0, and computing the euclidean distance to be the best method as $L2$ distance takes advantage of the higher dimensional space.

In the one-hot case, the model has to activate only 1 bit to deliver a classification. However, in the proposed method, the model has to activate K output bits to deliver

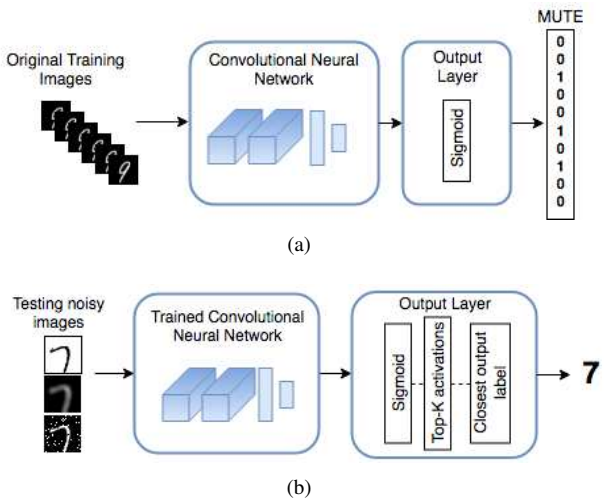

Figure 4. Overview of the (a) training and (b) inferencing methods

a classification. Even if one or two bits flip due to noise, it may not change the nearest label. Hence, MUTE can make a neural network robust enough to produce the correct prediction despite noise. Computing Euclidean distance between the set of target encoding may have a potential impact on inference latency. The latency could be minimized by optimizing the Euclidean distance computation using a fast XOR method.

4. Results & Discussion

4.1. Generation of MUTE

We compared our Narrow-and-converge approach with ILP optimization using CPLEX [1] in terms of the solution quality and computation time (Table 1). We broke-

down ILP optimization into two approaches: optimization with weights, and optimization without weights followed by shuffling with weights. The experiment was to generate $10-$bits with $4-$hot MUTE codes for 10 classes. Our experiment environment includes 56 CPU cores with 120GB RAM. CPLEX ran with parallel threads utilizing 32 cores in the experiment.

	Optimization approach	Obj. value	Min. HD	Compute time
$10-$bit $4-$hot	Narrow-and-converge	450.21	4	$90sec$
	CPLEX (w/ weights)	450.31	4	$> 11hrs$
	CPLEX (w/o weights)	447.75	4	$> 50hrs$

Table 1. The proposed Narrow-and-converge approach found the same quality solution, and reduced run-time by more than 99.75% compared to CPLEX. *HD: Hamming Distance*

For the $10-$bits/$4-$hot case with weights derived from MNIST dataset, Narrow-and-converge approach reduced the time cost by more than 99.75%, while it performs as good as ILP optimization. ILP with weighted objective function takes more than 11 hours to reach an objective value of 450.31. We were not able to run more than 11 hours due to memory limitations. Solving unweighted objective function for 50 hours followed by shuffling with weights reaches an objective value of 447.75.

Also, it took 46.3 minutes to generate MUTE for the $50-$bit/$20-$hot case ($N = 50$ and $K = 20$). It is possible to generate MUTE for hundreds of classes. But the memory and compute needed to run ILP even for a short duration with 1000s classes is prohibitive. The worst case time complexity of ILP is $\mathcal{O}(2^n)$. Due to limitations of ILP, we are unable to generate MUTE on datasets with a large number of output classes, such as ImageNet [6]. Yet, most classification systems deployed in the real-world are specialized systems that deal with hundreds of classes at most. Hence, we believe a robust encoding system such as MUTE is a useful method. Whereas, methods such as adversarial training to achieve robustness is not sustainable as the space of noisy data is much larger than that of clean data.

4.2. Performance of MUTE

Datasets. We used the MNIST [23], CIFAR-10 [19] and ICON-50 [13] datasets for experiments. MNIST is a dataset of handwritten digits. CIFAR-10 is a dataset of 32×32 color images belonging to 10 classes. ICON-50 is a set of $10,000$ color icons of size 32×32 belonging to 50 classes and collected from various companies. The icons from different companies exhibit different styles and versions. This is a challenging dataset because of style differences among icons collected from different companies, and class imbalance of icons from companies.

We used the standard train/test split for MNIST and CIFAR-10 datasets. For ICON-50, 80% of images in each class were randomly assigned to the train set and rest were allocated to the test set. In addition to the original test set, we created noisy and adversarial versions of the original test sets as shown in Figure 5 to evaluate the robustness of the trained models. Negative images [14] have the same spatial structure as the original images but are in a diagonally opposite color space. These images are useful to evaluate if the trained models have learned the semantic structure of the dataset or merely memorized the spatial pixel correlations in the training images. Typical noisy images such as Gaussian blurred images with $\sigma = 1$ and 2 and Salt & Pepper noise at 2% and 5% were created. We also created adversarial images with Fast Gradient Sign Method (FGSM) [11] with $\epsilon = 0.05, 0.1$ and 0.2. This creates a comprehensive test set that evaluates models for their generalization and robustness.

CNN Architectures. We utilized a wide range of CNN architectures in our experiments. MNIST dataset was trained and tested with two CNNs: LeNet [23] and ConvNet. ConvNet has 2 convolutional layers with 5×5 kernel size, followed by a fully connected layer with 50 neurons and a final sigmoid layer. CIFAR-10 and ICON-50 datasets are trained and tested with AlexNet [20], DenseNet [16], ResNet [12], and ResNeXt [34] architectures. DenseNet has depth of 40 and growth rate of 24. ResNet has depth of 20. ResNeXt has depth of 29 and cardinality of 8.

Experiments. The CNN architectures were trained with one-hot encoding, Hadamard encoding, and MUTE for different configurations of hot bits and weights on original images in the training datasets for 200 epochs starting from random initialization. The same set of hyper-parameters were used across the different target encodings to eliminate the impact of hyper-parameter choice on the model performance and for fair comparison. The trained models were then tested with original, noisy and adversarial images in the test set. We did not augment MNIST, however augmented CIFAR-10 and ICON-50 data using randomized affine transformations, color jitter and cropping in random order. The augmentation parameters were chosen by trial and error. The batch size was 128. CNNs were optimized using SGD with 0.9 momentum, 0.0001 weight decay, and 0.1 initial learning rate. All experiments were conducted on an Intel(R) machine with Xeon(R) CPU E5-2680 v4 at 2.40 GHz frequency. It has 504 GB RAM and 56 CPU cores. The machine has 4 Tesla P100-PCIE GPUs. Models were trained using only 1 GPU at a time.

The results of our experiments are shown in Figures 6, 7, 8, 9 and Table 2. In the legends, "MUTE $K-$hot" shows the proposed method with $K-$hot bits and without ambiguity weight generation by setting $W_{ij} = 1$ in Equation 1, and "MUTE Weighted $K-$hot" shows the proposed technique with $K-$hot bits and ambiguity weights generated to model the distance between the ambiguous classes.

In our experiments, we compared MUTE against the conventional one-hot encoding and the state-of-the-art Hadamard target encoding methods. Hadamard encod-

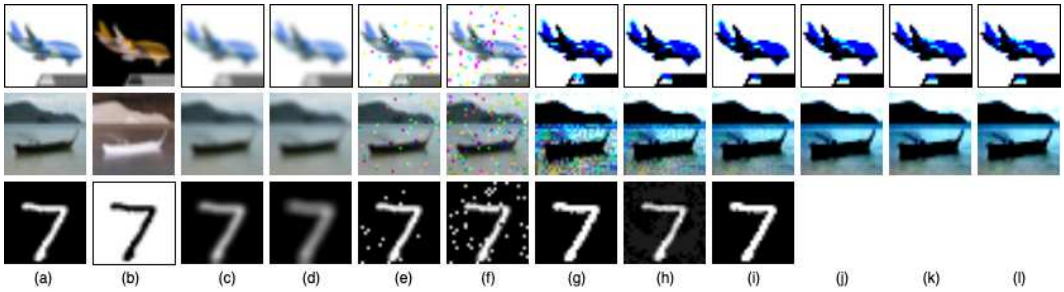

Figure 5. Images from the various test sets. The images in the first row is from the ICON-50 [13] dataset, second row is from the CIFAR-10 [19] dataset and the third row is from the MNIST [23] dataset. (a) Original images. (b) Negative [14] of the original images. (b - d) Original images blurred with Gaussian kernels of $\sigma = 1$ and $\sigma = 2$ respectively. (e - f) Salt & pepper noise added to the original images at 2% and 5% respectively. (g - l) Adversarial images with FGSM [11] noise with $\epsilon = 0.2, 0.1, 0.05, 0.01, 0.005$ and 0.001 respectively.

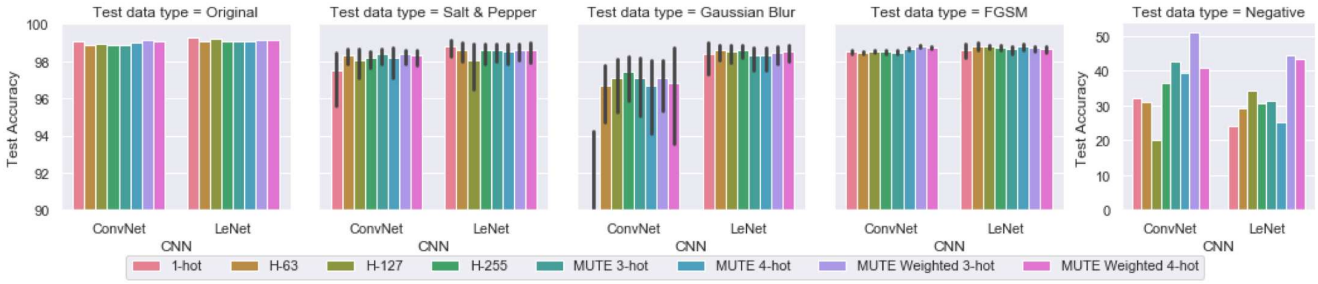

Figure 6. MUTE with LeNet and ConvNet architectures trained on MNIST data improves the one-hot average test accuracy by 2.8% and 7.1% respectively. Whereas Hadamard target encoding improves one-hot performance only by 1.7% and 3.5%.

ings (details in [35]), are derived from a Hadamard Matrix, $\mathcal{H} \in \{+1, -1\}^{m \times m}$ such that $\mathcal{H}\mathcal{H}^T = mI$ where I is an identity matrix. In this method, the model size is significantly increased by adapting to the target code lengths of 63, 127 or 255. Also, code assignment does not take inter-class ambiguity in to account. In addition to directly comparing with Hadamard encoding results in [35], we used Hadamard codes with our train and test methods. Hadamard results are shown as H-63, H-127 and H-255 in the Figures 6 and 7.

Figure 6 shows the test accuracy of LeNet and ConvNet architectures with different target encodings trained on original images in the MNIST training dataset and tested on original, noisy and adversarial versions of the MNIST test dataset. The barplots illustrate the central tendency for different test datasets and uncertainty (error bars) for test images impacted by varying amounts of Gaussian blur, Salt & Pepper noise and FGSM. The proposed MUTE method has better average test performance than one-hot encoding or Hadamard target encoding (H-63, H-127, and H-255). The best test accuracy on original images reported by [35] using Hadamard Codes on MNIST is 85.47% using direct classification with H-255. The authors [35] use a CNN architecture with 3 convolutional layers, 1 locally connected layer and 2 fully-connected layers. The proposed MUTE method improves this result by 13.61 percentage points on average.

For n classes, [18] uses n vectors of length l ($l = 2000$ for MNIST) whereas MUTE codes are n bits. Also, there is no notion of inter-class ambiguity in [18]. The model A_{RO} [18] tested with FGSM attacks of $= 0.2$ achieves 88.7% while various MUTE encodings achieve more than 98% accuracy.

CNN Arch.	Target Encoding				
	1-hot	MUTE			
		15-hot	20-hot	Weighted 15-hot	Weighted 20-hot
AlexNet	21.85	33.78	**34.45**	32.72	32.83
DenseNet	31.02	34.07	33.72	31.77	**34.95**
ResNet	29.87	32.31	31.56	32.21	**33.31**
ResNeXt	34.68	**34.86**	34.66	34.51	34.76

Table 2. Average test accuracy on noisy data of the ICON-50 subtype robustness experiment in [13].

CNN architectures with various MUTE were trained on original images in the CIFAR-10 training dataset and tested on original and noisy versions of the CIFAR-10 test dataset as shown in Figure 7. The proposed MUTE method with AlexNet, DenseNet, ResNet and ResNeXt architectures improved average test accuracy over one-hot encoding. However, choice of architecture seems to be important when using Hadamard encodings, as performance dropped when using DenseNet and ResNeXt architectures. MUTE is particularly useful when a less robust architecture such as AlexNet [13] is used.

We trained AlexNet, DenseNet, ResNet, and ResNeXt architectures on an ICON-50 training set that contained images from all 50 classes and tested on original and noisy images in the test set (Figure 8). We did not include Hadamard coding for ICON-50 as it was not provided by the original paper. MUTE increased one-hot encoding performance 0.3% to 42.29% with AlexNet, the least robust CNN [13],

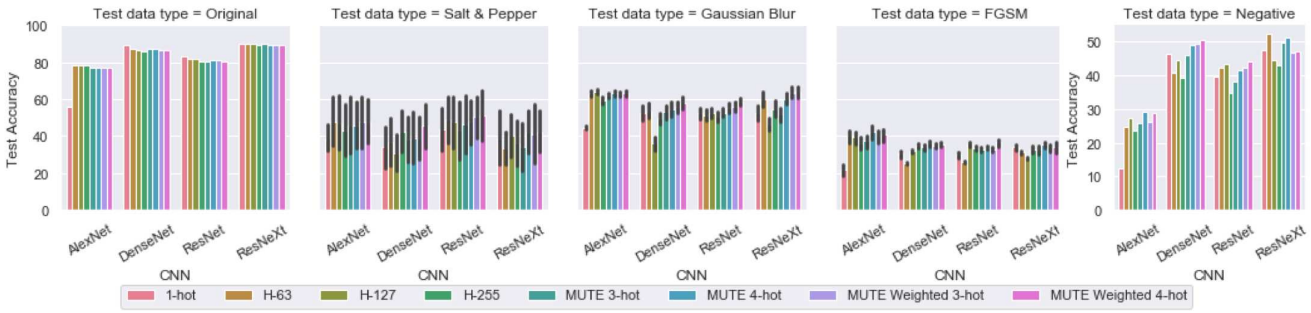

Figure 7. MUTE encoding with AlexNet, DenseNet, ResNet and ResNeXt architectures trained on CIFAR-10 data improves the one-hot average test accuracy by 41.5% , 4.2%, 3.6% and 3.8% respectively. Whereas Hadamard target encoding improves one-hot performance only when used with AlexNet and ResNet architecture. When used with DenseNet and ResNeXt, the average test accuracy is worse than using one-hot encoding.

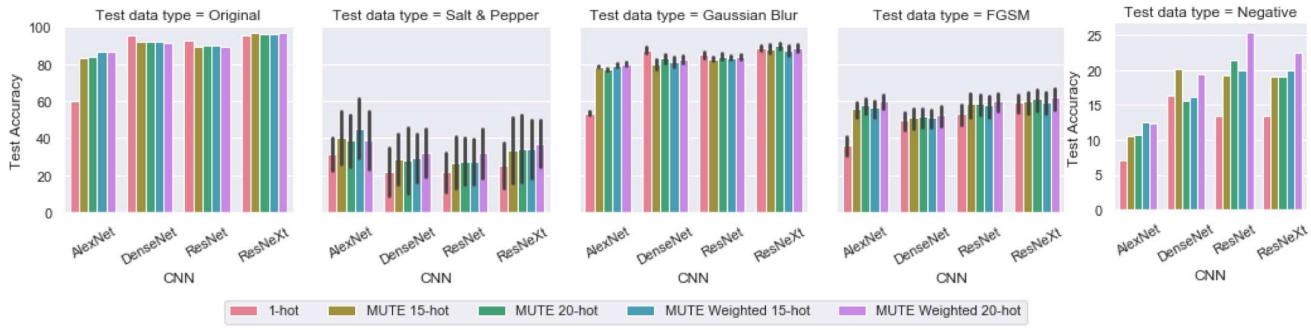

Figure 8. AlexNet, DenseNet, ResNet, and ResNeXt architectures trained with various MUTE on ICON-50 data and tested with the original and noisy testsets improved average test accuracy over one-hot by 42.29%, 0.37%, 4.70%, and 8.12% respectively.

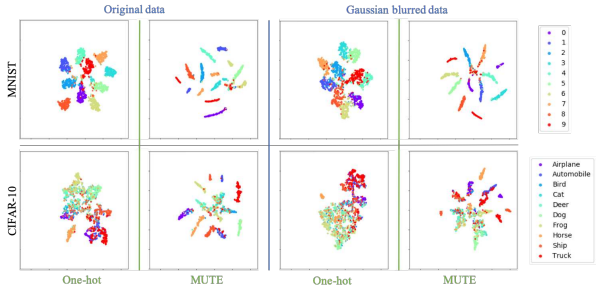

Figure 9. Features learned with MUTE are well-separated and condensed in multi-dimensional space such that model maintains separation even when tested with noisy data. In comparison, features learned by one-hot encodings lack isolation in feature space.

benefiting the most from MUTE. In another experiment using ICON-50 dataset, we trained the CNN architectures on all original images belonging to the sub-type robustness experiment in [13]. Then tested on the noisy versions (negative, Gaussian blurred, and Salt & Pepper noise) of the held-out sub-types. Table 2 lists the average test accuracy on the noisy test sets. MUTE consistently obtains higher average test accuracy for any number of hot bits or weighted configuration. We used the TSNE [25] algorithm to visualize ConvNet and AlexNet models trained with one-hot and MUTE on MNIST and CIFAR-10 datasets, respectively. Figure 9 shows that models trained with one-hot encodings fail to learn distinguishable features that give rise to good decision boundaries. Thus, noisy data is easily mis-predicted. On the other-hand, condensed features learned by MUTE enable clear decision boundaries even when faced with noisy data.

MUTE improved average test accuracy on various CNN architectures in the range of 42.29% to 0.37% (median = 4.45%). All experiment timing were recorded. However, there was no statistically significant difference between one-hot and MUTE train and inference times. We propose to use MUTE during the design phase when it is hard to gauge if a CNN architecture has sufficient learning capacity for a given task. A key benefit of using MUTE is that it would extract more robust features and better utilize the given learning capacity in the CNN. As seen in Figure 6, the performance of MUTE relative to one-hot encoding is more stronger on a weaker CNN such as ConvNet, than on a stronger CNN as LeNet. When a CNN has weak structure but has sufficient learning capacity, MUTE better utilizes the excess neurons to learn more predictive features than one-hot encoding. Even with limited number of neurons (e.g., ConvNet), MUTE still delivers the better accuracy and robustness than comparative methods. Overall, our results are encouraging because MUTE learns robust features for any given CNN architecture which could lead to a smaller, thus faster model at the end of CNN design exploration.

References

[1] Cplex. https://www.ibm.com/analytics/cplex-optimizer. Accessed: November 14, 2019.

[2] Zeynep Akata, Florent Perronnin, Zaid Harchaoui, and Cordelia Schmid. Label-embedding for image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(7):1425–1438, 2016.

[3] Yonatan Amit, Michael Fink, Nathan Srebro, and Shimon Ullman. Uncovering shared structures in multiclass classification. In *Proceedings of the 24th international conference on Machine learning*, pages 17–24. ACM, 2007.

[4] Moustapha M Cisse, Nicolas Usunier, Thierry Artieres, and Patrick Gallinari. Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*, pages 1851–1859, 2013.

[5] Huiqun Deng, George Stathopoulos, and Ching Y Suen. Applying error-correcting output coding to enhance convolutional neural network for target detection and pattern recognition. In *2010 20th International Conference on Pattern Recognition*, pages 4291–4294. IEEE, 2010.

[6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[7] Thomas G Dietterich and Ghulum Bakiri. Solving multiclass learning problems via error-correcting output codes. *Journal of artificial intelligence research*, 2:263–286, 1994.

[8] Charles M Fiduccia and Robert M Mattheyses. A linear-time heuristic for improving network partitions. In *19th Design Automation Conference*, pages 175–181. IEEE, 1982.

[9] Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *Advances in neural information processing systems*, pages 2121–2129, 2013.

[10] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna M. Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *CoRR*, 2018.

[11] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[13] Dan Hendrycks and Thomas G Dietterich. Benchmarking neural network robustness to common corruptions and surface variations. *arXiv preprint arXiv:1807.01697*, 2018.

[14] Hossein Hosseini, Baicen Xiao, Mayoore Jaiswal, and Radha Poovendran. On the limitation of convolutional neural networks in recognizing negative images. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 352–358. IEEE, 2017.

[15] Daniel J Hsu, Sham M Kakade, John Langford, and Tong Zhang. Multi-label prediction via compressed sensing. In *Advances in neural information processing systems*, pages 772–780, 2009.

[16] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[17] Inseok Hwang, Jinho Lee, Frank Liu, and Minsik Cho. Simex: Express prediction of inter-dataset similarity by a fleet of autoencoders. *arXiv preprint arXiv:2001.04893*, 2020.

[18] Donghyun Kim, Sarah Adel Bargal, Jianming Zhang, and Stan Sclaroff. Multi-way encoding for robustness to adversarial attacks, 2019.

[19] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[21] Ludmila I Kuncheva. Using diversity measures for generating error-correcting output codes in classifier ensembles. *Pattern Recognition Letters*, 26(1):83–90, 2005.

[22] John Langford and Alina Beygelzimer. Sensitive error correcting output codes. In *International Conference on Computational Learning Theory*, pages 158–172. Springer, 2005.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[24] Yin Li, Xiaodi Hou, Christof Koch, James M Rehg, and Alan L Yuille. The secrets of salient object segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 280–287, 2014.

[25] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[26] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[27] Pau Rodríguez, Miguel A Bautista, Jordi Gonzalez, and Sergio Escalera. Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75:21–31, 2018.

[28] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[29] Tatiana Tommasi, Novi Patricia, Barbara Caputo, and Tinne Tuytelaars. A deeper look at dataset bias. In *Domain Adaptation in Computer Vision Applications*, pages 37–55. Springer, 2017.

[30] Antonio Torralba, Alexei A Efros, et al. Unbiased look at dataset bias. In *CVPR*, volume 1, page 7. Citeseer, 2011.

[31] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of machine learning research*, 6(Sep):1453–1484, 2005.

[32] J. R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1):31–42, Jan. 1976.

[33] Jason Weston, Samy Bengio, and Nicolas Usunier. Large scale image annotation: learning to rank with joint word-image embeddings. *Machine learning*, 81(1):21–35, 2010.

[34] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500, 2017.

[35] Shuo Yang, Ping Luo, Chen Change Loy, Kenneth W Shum, and Xiaoou Tang. Deep representation learning with target coding. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[36] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.