Error-Correcting Output Codes with Ensemble Diversity for Robust Learning in Neural Networks

Yang Song*, Qiyu Kang*, and Wee Peng Tay
Nanyang Technological University, 50 Nanyang Ave, Singapore 639798
songy@ntu.edu.sg, kang0080@e.ntu.edu.sg, wptay@ntu.edu.sg

Abstract

Though deep learning has been applied successfully in many scenarios, malicious inputs with human-imperceptible perturbations can make it vulnerable in real applications. This paper proposes an error-correcting neural network (ECNN) that combines a set of binary classifiers to combat adversarial examples in the multi-class classification problem. To build an ECNN, we propose to design a code matrix so that the minimum Hamming distance between any two rows (i.e., two codewords) and the minimum shared information distance between any two columns (i.e., two partitions of class labels) are simultaneously maximized. Maximizing row distances can increase the system fault tolerance while maximizing column distances helps increase the diversity between binary classifiers. We propose an end-to-end training method for our ECNN, which allows further improvement of the diversity between binary classifiers. The end-to-end training renders our proposed ECNN different from the traditional error-correcting output code (ECOC) based methods that train binary classifiers independently. ECNN is complementary to other existing defense approaches such as adversarial training and can be applied in conjunction with them. We empirically demonstrate that our proposed ECNN is effective against the state-of-the-art white-box and black-box attacks on several datasets while maintaining good classification accuracy on normal examples.

Introduction

Deep learning has been widely and successfully applied in many tasks such as image classification (Krizhevsky, Sutskever, and Hinton 2012; LeCun et al. 1998), speech recognition (Hinton et al. 2012), and natural language processing (Andor et al. 2016). However, recent works (Szegedy et al. 2013) showed that the original images can be modified by an adversary with human-imperceptible perturbations so that the deep neural networks (DNNs) are fooled into mis-classifying them. To mitigate the effect of adversarial attacks, many defense approaches have been proposed. Generally speaking, they fall into three categories:

1. adversarial training, which augments the training data with adversarial examples (Szegedy et al. 2013; Goodfellow, Shlens, and Szegedy 2015),
2. modifying the DNN or training procedure, e.g., defensive distillation (Papernot et al. 2016b), and
3. post-training defenses, which attempt to remove the adversarial noise from the input examples (Hendrycks and Gimpel 2017; Meng and Chen 2017; Samangouei, Kabkab, and Chellappa 2018) in the testing phase.

In this paper, we propose an approach for the second category. Most defense approaches of this type focus on robustifying a single network, while a few works have adopted ensemble methods (Abbasi and Gagné 2017; Xu, Evans, and Qi 2018; Pang et al. 2019; Sen, Ravindran, and Raghunathan 2020). These ensemble methods build a new classifier consisting of several base classifiers that are assigned with the same classification task. Promoting the diversity among the base classifiers during training is essential to prevent adversarial examples from transferring between them, since the adversarial examples crafted for one classifier may also fool the others.

The use of error correcting output codes (ECOC) (Dietterich and Bakiri 1994; García-Pedrajas and Fyfe 2008) differs from the above mentioned ensemble methods by assigning each class an unique codeword. This forms a pre-defined code matrix. For an illustration, see Table 1, which shows an example of a code matrix for three classes with each class being represented by four bits. Each column splits the original classes into two meta-classes, meta-class '0' and meta-class '1'. A binary classifier is then learned independently for each column of the code matrix. To classify a new sample, all binary classifiers are evaluated to obtain a binary string. Finally, the method assigns the class whose codeword is closest to the obtained binary string to the sample. In the literature, ECOC is mostly used with decision trees or shallow neural networks.

Class	Net 0	Net 1	Net 2	Net 3
0	1	0	1	0
1	1	1	0	1
2	0	0	0	1

Table 1: Example of a 3×4 code matrix.

In this paper, we utilize the concept of ECOC in a deep neural network, which we call error-correcting neural net-

*Two authors contributed equally to this work.

work (ECNN). In traditional ECOC, a code matrix is generated in such a way that the minimum Hamming distance between any two rows is maximized. Maximizing the row distance creates sufficient redundant error-correcting bits, thus enhancing the classifier's error-tolerant ability. However, it is possible for an adversary to design adversarial examples that trick most of the binary classifiers since they are not fully independent of each other. Therefore, to mitigate this effect, the code matrix should be designed so that the binary tasks are as different from each other as possible. When designing a code matrix for ECNN, we attempt to separate columns (binary tasks) by maximizing the minimum shared information distance (Meilă 2003) between any two columns. Maximizing the column distance inherently promotes the diversity between binary classifiers. This is essential to prevent the adversarial examples crafted for one binary classifier transferring to the other binary classifiers.

After our preliminary work, we became aware of a recent independent work (Verma and Swami 2019), which designs a DNN using error-correcting codes and shares similar concepts with our proposed approach. The main difference to our work is that our encoder, i.e., all the binary classifiers are trained jointly whereas (Verma and Swami 2019) splits the binary classifiers into several groups, with each being trained separately. The work (Verma and Swami 2019) only forces a pre-training diversity using its code matrix whereas ECNN further includes a diversity promoting regularizer during training. This novelty improves the testing accuracy of ECNN compared to (Verma and Swami 2019).

During training of ECNN, all binary classifiers are jointly trained and their outputs, i.e., the predicted meta-class classification probabilities, are concatenated and fed into a decoder to obtain the predicted classification probabilities. We propose an end-to-end training method that allows for exploiting the interaction between binary classifiers, thus further improving the diversity between them. Our main contributions are summarized as follows:

1. We apply error-correcting codes to build a DNN for classification and propose an end-to-end training method. For illustration, we focus our discussion on the problem of robust image classification.
2. We provide theoretical analysis that helps to guide the design of ECNN, including the choice of activation functions.
3. In our experiments, we test ECNN on several widely used datasets MNIST (LeCun, Corte, and Burges 2010), CIFAR-10 and CIFAR-100 (Krizhevsky and Hinton 2009) under several well-known adversarial attacks. We demonstrate empirically that ECNN is robust against adversarial white-box attacks with improvement in classification accuracy of adversarial examples of up to 14.8 and 17.4 percentage points compared to another current state-of-the-art ensemble method (Verma and Swami 2019) on MNIST and CIFAR-10, respectively, while ECNN uses 22.2% more parameters than (Verma and Swami 2019) on MNIST and 78.8% *less* parameters than (Verma and Swami 2019) on CIFAR-10.
4. We also test ECNN on the German Traffic Sign Recognition Benchmark (GTSRB) (Stallkamp et al. 2012) under black-box setting and show that ECNN outperforms the baseline on both normal and adversarial examples.
5. When combined with adversarial training, ECNN further improves its robustness, with improvement in correct classification of adversarial examples by about 18 percentage points compared to pure adversarial training.

Notation	Definition
N	number of binary classifiers
$g_{\theta_n}(\cdot)$	feature extraction function at the n-th classifier
$\mathbf{f}_n$	feature vector $\mathbf{f}_n = g_{\theta_n}(x)$
F	dimension of feature vector $\mathbf{f}_n$
$h_n(\cdot)$	prediction function at the n-th binary classifier
z_n	encoder's outputs $z_n = h_n(\mathbf{f}_n)$
$\boldsymbol{\phi}_n$	linear form of h_n, i.e., $z_n = \boldsymbol{\phi}_n \mathbf{f}_n$
K	number of input samples $x \in \{x_0, \ldots, x_{K-1}\}$
$\mathbf{f}_{n,x}$	$\mathbf{f}_n$ depending on input sample x
$\mathbf{f}_n^{y(x)}$	principal feature vector associated with class $y(x)$
$\mathbf{n}_{n,x}$	random perturbation $\mathbf{f}_{n,x} = \mathbf{f}_n^{y(x)} + \mathbf{n}_{n,x}$
$\nu(\cdot)$	logistic function
$\sigma(\cdot)$	softmax function
$y_n(x)$	meta-class $y_n(x) = \mathbf{M}(y(x), n)$

Table 2: Summary of commonly-used symbols.

Furthermore, we show how to generalize the binary classifiers used in ECNN to q-ary classifiers using a q-ary code matrix.

The rest of this paper is organized as follows. Firstly, we present our ECNN framework, its training strategy and some theoretical analysis of its properties. Then, we present extensive experimental results, and we conclude in the last section. We refer interested readers to the supplementary material for a more detailed account of several recent works that defend against adversarial examples using ensembles of models (Abbasi and Gagné 2017; Xu, Evans, and Qi 2018; Pang et al. 2019; Verma and Swami 2019; Sen, Ravindran, and Raghunathan 2020) and some popular adversarial attacks (Goodfellow, Shlens, and Szegedy 2015; Kurakin, Goodfellow, and Bengio 2017; Mądry et al. 2018; Papernot et al. 2016a; Carlini and Wagner 2017) that are used to verify the robustness of our proposed ECNN. The proofs for all lemmas in this paper are given in the supplementary material. For easier reference, we summarize some of the commonly-used symbols in Table 2.

Error-Correcting Neural Network

In this section, we present the architectures of the encoder and the decoder in ECNN, the training strategy and the way to generate the code matrix. We provide theoretical results that help to guide us in the design of ECNN. Finally, we show how to extend to q-ary classifiers in the encoder, where $q > 2$.

The overall architecture of ECNN is shown in Fig. 1. Consider a M-ary classification problem. A $M \times N$ binary code matrix $\mathbf{M}$, where $N \geq 1$, encodes each class with a N-bit codeword. For each $n = 0, \ldots, N-1$, the n-th bits of all the

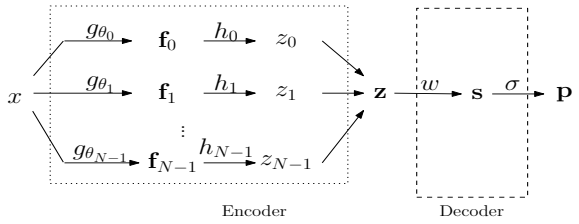


Fig. 1: ECNN architecture.

M codewords define a binary meta-classification problem, with $y_n(x) = \mathbf{M}(y(x), n) \in \{0, 1\}$ being the meta-class of sample x if $y(x)$ is the label of x. For example, in the code matrix shown in Table 1, the first bits or "Net 0" as indicated in the table correspond to a binary classification problem that distinguishes classes 0 or 1 from class 2. We learn a binary classifier corresponding to each binary meta-classification problem and combine their outputs together. The collection of these binary classifiers is called the *encoder* in our architecture. We call each binary classifier in the encoder a *meta classifier*. The encoder is then followed by a *decoder* whose function is to infer which of the M classes the sample belongs to. In the following, we describe both the encoder and decoder in detail. Let $y(x)$ be the label of sample x. We suppose that a training set $\{(x_k, y(x_k)) : k = 0, \ldots, K-1\}$ is available. In our discussions, we append a subscript x_k to a quantity (e.g., $\mathbf{f}_{n,x_k}$ in place of $\mathbf{f}_n$) if we wish to emphasize its dependence on the sample x_k.

Encoder

The encoder contains N composite functions $\sigma \circ h_n \circ g_{\theta_n}$, for $n = 0, \ldots, N-1$, each corresponding to a binary classifier. The function g_{θ_n} extract feature vectors $\mathbf{f}_n = g_{\theta_n}(x) \in \mathbb{R}^F$ from the input x, h_n is a prediction function that outputs $z_n = h_n(\mathbf{f}_n) \in \mathbb{R}^1$, which can be interpreted as probabilities after normalization.

We choose h_n to be a simple linear function, i.e., $z_n = \boldsymbol{\phi}_n \mathbf{f}_n$, for $n = 0, \ldots, N-1$. Furthermore, in our final architecture, we set $\boldsymbol{\phi}_n = \boldsymbol{\phi}$ for all $n = 0, \ldots, N-1$. The reason for our choice is explained later in the next subsection where we introduce the concept of ensemble diversity. Let $\mathbf{M} \in \{0, 1\}^{M \times N}$ be the given code matrix. Denote the loss function for the encoder as $\ell(z_n, y_n(x))$. The encoder can be formulated as an optimization problem:

$$\min_{\{\theta_n, \boldsymbol{\phi}_n\}_{n=0}^{N-1}} \frac{1}{N(N-1)K} \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} \ell(z_{n,x_k}, y_n(x_k)) \quad (1)$$
$$\text{s.t. for } k = 0, \ldots, K-1 \text{ and } n = 0, \ldots, N-1,$$
$$z_{n,x_k} = \boldsymbol{\phi}_n g_{\theta_n}(x_k).$$

The loss function $\ell(z_n, y_n(x))$ would be

$$\max(0, 1 - z_n(2y_n(x) - 1)), \text{ for hinge loss,} \quad (2)$$
$$-\mathbf{1}_{y_n(x)}^{\intercal} \log(\boldsymbol{\zeta}_n), \text{ for cross entropy loss,} \quad (3)$$

where $\mathbf{1}_{y_n(x)}$ is the one-hot encoding of the meta-class $y_n(x)$, i.e., a vector with 1 at the $y_n(x)$-th entry and 0 for all the other entries and $\boldsymbol{\zeta}_n = [1 - \nu(z_n), \nu(z_n)]^{\intercal}$ with $\nu(\cdot)$ being the logistic function. Finally, the logits z_n are concatenated into $\mathbf{z} = [z_0, \ldots, z_{N-1}]^{\intercal}$, which is the input to the decoder in ECNN.

Ensemble Diversity

To mitigate against adversarial attacks, each feature vector $\mathbf{f}_n$, $n = 0, \ldots, N-1$, should only be instrumental in its own binary classifier's prediction while being insensitive to the other binary classifiers' predictions. However, it is difficult to directly define a difference measurement between features. One possible way is to ensure linear independence between them and measure the independence using singular values. This operation is computationally costly, making it unsuitable for training in neural networks. Furthermore, even if feature vectors are linearly independent, it may very well happen that $\mathbf{f}_i$ is just a permutation of $\mathbf{f}_j$ for $i \neq j$. The use of L_p distance to measure differences in feature vectors is therefore inappropriate. We choose to promote ensemble diversity in terms of the output probability of each binary classifier by solving the following optimization problem:

$$\max_{\{\boldsymbol{\phi}_i\}_{i=0}^{N-1}} \frac{1}{N(N-1)K} \sum_{k=0}^{K-1} \sum_{i \neq j} -[1 - \nu(\boldsymbol{\phi}_i \mathbf{f}_{j,x_k}), \nu(\boldsymbol{\phi}_i \mathbf{f}_{j,x_k})] \log([1 - \nu(\boldsymbol{\phi}_i \mathbf{f}_{j,x_k}), \nu(\boldsymbol{\phi}_i \mathbf{f}_{j,x_k})]^{\intercal}), \quad (4)$$
$$\text{s.t. } \mathbf{f}_{j,x_k} = g_{\theta_j}(x_k),\ j = 0, \ldots, N-1,$$
$$k = 0, \ldots, K-1.$$

Assuming that the input x is drawn from a distribution, let $\mathbf{f}_n^y$ be the expected feature vector of class y generated by the n-th classifier in the encoder. We call this the principal feature vector of class y. Then for any input x, we have

$$\mathbf{f}_{n,x} = \mathbf{f}_n^{y(x)} + \mathbf{n}_{n,x}, \quad (5)$$

where $\mathbf{n}_{n,x}$ is a zero-mean random perturbation. We make the following assumption.

Assumption 1. *The random vectors $\{\mathbf{n}_{n,x_k} : k = 0, \ldots, K-1\}$ have a joint distribution absolutely continuous with respect to (w.r.t.) Lebesgue measure.*

The above assumption is satisfied if the training samples $\{x_k : k = 0, \ldots, K-1\}$ are drawn independent and identically distributed (i.i.d.) from a distribution absolutely continuous w.r.t. Lebesgue measure.

Lemma 1. *Suppose that $\{\theta_n\}_{n=0}^{N-1}$ satisfy Assumption 1. Then the following statements hold with probability one:*

(a) Suppose $\theta_n = \theta$ for all $n = 0, \ldots, N-1$. There exists $\{\boldsymbol{\phi}_n : n = 0, \ldots, N-1\}$ such that the loss of (1) is arbitrarily small if $K \leq F$.

(b) Suppose $\boldsymbol{\phi}_n = \boldsymbol{\phi}$ for all $n = 0, \ldots, N-1$. There exists a $\boldsymbol{\phi}$ such that the loss of (1) is arbitrarily small if $NK \leq F$.

(c) Suppose $N > 1$. Any feasible solution of (1) cannot have $\theta_n = \theta$ and $\boldsymbol{\phi}_n = \boldsymbol{\phi}$ for some θ and $\boldsymbol{\phi}$, and for all $n = 0, \ldots, N-1$.

Lemma 2. *Suppose that $\{\theta_n\}_{n=0}^{N-1}$ satisfy Assumption 1. Suppose further that $\{\theta_n\}_{n=0}^{N-1}$ are chosen so that for each $n = 0, \ldots, N-1$ and $y = 0, \ldots, M-1$, $\mathbf{f}_n^y = \mathbf{S}_n \mathbf{f}^y$ for some $\mathbf{f}^y \in \mathbb{R}^F$, where $\mathbf{S}_n$ is a permutation matrix.*

(a) There exist linear transformations $\{\boldsymbol{\phi}_n\}_{n=0}^{N-1}$ such that the loss of (1) *is arbitrarily small.*

(b) If $NM \geq F(M+1)$ and $\boldsymbol{\phi}_n$ are constrained to be the same for all $n = 0, \ldots, N-1$, then for almost every (w.r.t. Lebesgue measure) $\{\mathbf{f}^y\}_{y=0}^{M-1}$, there is no feasible solution to (1)*.*

Lemma 3. *Suppose that $\{\theta_n\}_{n=0}^{N-1}$ satisfy Assumption 1, and $\{\mathbf{f}_n^y : y = 0, \ldots, M-1, n = 0, \ldots, N-1\}$ are linearly independent. For all $n = 0, \ldots, N-1$, $\boldsymbol{\phi}_n$ are constrained to be the same. Then, the n-th binary classifier in the encoder classifies the sample x_k, $k = 0, \ldots, K-1$, correctly if $\|\mathbf{n}_{n,x_k}\|_2$ is sufficiently small.*

Lemma 1 shows that if we constrain either the parameters θ_n or transforms $\boldsymbol{\phi}_n$ (but not both) in the encoder to be identical across binary classifiers, then it is still possible to achieve arbitrarily small loss. Lemma 2 suggests that if we do not constrain the transforms $\boldsymbol{\phi}_n$ to be the same, then optimizing (1) may produce a solution where the features $\mathbf{f}_n^{y(x)}$ for different binary classifiers $n = 0, \ldots, N-1$ are permutations of each other for the same sample x. This is clearly undesirable as any adversarial attack on a particular binary classifier can then translate to the other classifiers. On the other hand, Lemma 3 suggests that if we constrain the transforms $\boldsymbol{\phi}_n$ to be the same, and choose the parameters θ_n to make $\{\mathbf{f}_n^y : y = 0, \ldots, M-1, n = 0, \ldots, N-1\}$ linearly independent, then high classification accuracy is still achievable if the input sample does not deviate too much from the mean. Our results thus suggest that a good strategy is to share the transform $\boldsymbol{\phi}$ across all the prediction functions. Then, (4) becomes

$$\begin{aligned} \max_{\boldsymbol{\phi}} \ & \frac{1}{NK} \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} -\boldsymbol{\zeta}_{n,x_k}^{\intercal} \log(\boldsymbol{\zeta}_{n,x_k}), \qquad (6) \\ \text{s.t. for } & n = 0, \ldots, N-1,\ k = 0, \ldots, K-1, \\ & \boldsymbol{\zeta}_{n,x_k} = \left[1 - \nu\left(\boldsymbol{\phi}\mathbf{f}_{n,x_k}\right), \nu\left(\boldsymbol{\phi}\mathbf{f}_{n,x_k}\right)\right]^{\intercal}, \\ & \mathbf{f}_{n,x_k} = g_{\theta_n}(x_k). \end{aligned}$$

Joint Optimization

The joint optimization that combines (1) and (6) can be formulated as

$$\begin{aligned} \min_{\{\theta_n\}_{n=0}^{N-1}, \boldsymbol{\phi}} \ & \frac{1}{NK} \sum_{k=0}^{K-1} \sum_{n=0}^{N-1} \ell(z_{n,x_k}, y_n(x_k)) \\ & - \gamma \boldsymbol{\zeta}_{n,x_k} \log\left(\boldsymbol{\zeta}_{n,x_k}\right), \qquad (7) \\ \text{s.t. for } & n = 0, \ldots, N-1,\ k = 0, \ldots, K-1, \\ & z_{n,x_k} = \boldsymbol{\phi} g_{\theta_n}(x_k), \end{aligned}$$

where $\gamma \geq 0$ is a configurable weight. The joint optimization (7) enables end-to-end training for ECNN.

Note that if we use cross entropy for $\ell(z_n, y_n(x))$, we convert one-hot labels $\mathbf{1}_{y_n(x_k)}$ to soft labels $\mathbf{1}_{y_n(x_k)} - \gamma\boldsymbol{\zeta}_{n,x_k}$ for all n, k by optimizing (7). This is also known as label smoothing (Szegedy et al. 2016), which is beneficial in improving the adversarial robustness in ECNN. The following Lemma 4 guides us in the choice of γ given the smoothed probabilities $\{\boldsymbol{\zeta}_{n,x_k}(y_n(x_k)) : n = 0, \ldots, N-1, k = 0, \ldots, K-1\}$.

Lemma 4. *The optimal solution of* (7)*, where we use cross entropy for $\ell(z_n, y_n(x))$, satisfies $\frac{1}{\boldsymbol{\zeta}_{n,x_k}(y_n(x_k))} = \gamma \log \frac{\boldsymbol{\zeta}_{n,x_k}(y_n(x_k))}{1 - \boldsymbol{\zeta}_{n,x_k}(y_n(x_k))}$ for all $n = 0, \ldots, N-1, k = 0, \ldots, K-1$.*

Decoder

The decoder involves a simple comparison with the code matrix $\mathbf{M}$. To enable the use of back-propagation, our decoder uses continuous relaxation: we compute the correlation between the real-valued string $\tanh(z)$ and each class's codeword (after scaling $[-1, 1]$), and the class having the maximum correlation is assigned as the output label. Mathematically, the decoder is a composite function $\sigma \circ w$. The function w first scales the logits $\mathbf{z}$ to $[-1, 1]$ using $\tanh$ function and then computes the inner products between the scaled logits and each row in $2\mathbf{M} - \mathbb{1}$, i.e., $\mathbf{s} = w(\tanh(\mathbf{z})) = (2\mathbf{M} - \mathbb{1})\tanh(\mathbf{z}) \in \mathbb{R}^M$, where $\mathbb{1}$ denotes a matrix with all its entries being 1. The $\sigma(\cdot)$ is the softmax function which maps $\mathbf{s}$ to the prediction probabilities $\mathbf{p} = \sigma\left((2\mathbf{M} - \mathbb{1})\tanh(\mathbf{z})\right) \in \mathbb{R}^M$.

Code Matrix Design

Let $\mathbf{r}_i$ be the i-th codeword of code matrix $\mathbf{M}$, and $H(\mathbf{r}_i, \mathbf{r}_j)$ be the Hamming distance between the i-th and j-th codewords of the code matrix, i.e., the number of bit positions where they differ. The minimum Hamming distance of code matrix $\mathbf{M}$ is then given by $d_H(\mathbf{M}) = \min_{i \neq j} H(\mathbf{r}_i, \mathbf{r}_j)$. A code matrix with larger $d_H(\mathbf{M})$ is preferred since it can correct more errors, resulting in better classification performance. However, if we only consider maximizing $d_H(\mathbf{M})$ when designing $\mathbf{M}$, the two columns of the matrix may lead to the corresponding binary classifiers performing the same classification task even though they have different bits. For a concrete illustration, consider again the code matrix example in Table 1. The last two columns in Table 1 are different, while the corresponding "Net 2" and "Net 3" are essentially performing the same task: classifying the classes to set $\{0\}$ or $\{1, 2\}$. An adversarial example generated by an untargeted attack that fools "Net 2" will also fool "Net 3". To further promote the diversity between binary classifiers, we therefore include column diversity when designing the code matrix. Specifically, we view each column of the code matrix as partitioning the M classes into clusters, and measure the difference between two columns using the variation of information (VI) metric (Meilă 2003). VI, which is a criterion for comparing the difference between two binary partitions, measures the amount of information lost and gained in changing from one clustering to another clustering (Meilă 2003). Each column of the ECNN code matrix can be interpreted to be a binary cluster. We denote the n-th column of code matrix $\mathbf{M}$ as $\mathbf{c}_n$, a set of classes that belong to meta-class k in the n-th column as C_n^k, where $n = 0, \ldots, N-1$ and $k = 0, 1$. The VI distance defined in (Meilă 2003) between $\mathbf{c}_m$ and $\mathbf{c}_n$, i.e., $\mathrm{VI}(\mathbf{c}_m, \mathbf{c}_n)$, is

$$-\sum_{k=0}^{1} s_m(k) \log s_m(k) - \sum_{k'=0}^{1} s_n(k') \log s_n(k')$$

$$-2\sum_{k=0}^{1}\sum_{k'=0}^{1} s_{m,n}(k,k')\log\frac{s_{m,n}(k,k')}{s_m(k)s_n(k')}, \tag{8}$$

where $s_{m,n}(k,k') = \frac{\left|C_m^k \cap C_n^{k'}\right|}{M}$, $s_m(k) = \frac{|C_m^k|}{M}$, $s_n(k') = \frac{\left|C_n^{k'}\right|}{M}$, and $|A|$ denotes the cardinality of the set A. The minimum VI distance of code matrix $\mathbf{M}$ is then given by $d_{\mathrm{VI}}(\mathbf{M}) = \min_{m\neq n}\mathrm{VI}(\mathbf{c}_m, \mathbf{c}_n)$.

The code matrix is designed to $\max_{\mathbf{M}} d_H(\mathbf{M}) + d_{\mathrm{VI}}(\mathbf{M})$, which is however a NP-complete problem (Pujol, Radeva, and Vitria 2006). We adopt simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983; Gamal et al. 1987) to solve this optimization problem heuristically, where the energy function is set as:

$$\min_{\mathbf{M}} \sum_{i\neq j} H(\mathbf{r}_i, \mathbf{r}_j)^{-2} + \eta \sum_{m\neq n} \mathrm{VI}(\mathbf{c}_m, \mathbf{c}_n)^{-2}, \tag{9}$$

and η is chosen such that the two summations are roughly equally weighted. This is a common technique used in simulated annealing (Gamal et al. 1987) as bit changes not involving the minimum distance pairs are not reflected in the energy function if it is set as $\max_{\mathbf{M}} d_H(\mathbf{M}) + d_{\mathrm{VI}}(\mathbf{M})$.

q-ary ECNN

A natural extension is to use a general q-ary code matrix with each meta classifier performing a q-ary classification problem. Since $\max d_H(\mathbf{M})$ for a q-ary $\mathbf{M}$ is no less than $\max d_H(\mathbf{M})$ for a binary $\mathbf{M}$, using a q-ary $\mathbf{M}$ with $q > 2$ has better error-correcting capacity than using a binary $\mathbf{M}$. Hence, better classification accuracy on normal images is expected as q increases. On the other hand, we show in Lemma 5 below that more information about the original classes is revealed in each q-ary classifier as q increases, thus rendering less adversarial robustness.

Lemma 5. *Suppose* $\mathbf{y} = [0, \ldots, M-1]^\intercal$ *and* $\mathbf{c}$ *is a column of a* q*-ary code matrix* $\mathbf{M}$*. The mutual information between* $\mathbf{y}$ *and* $\mathbf{c}$ *can be defined, analogously to* (8)*, as*

$$I(\mathbf{y}, \mathbf{c}) = \sum_{k=0}^{q-1}\sum_{\ell=0}^{M-1} s(k,\ell)\log\frac{s(k,\ell)}{s(k)s(\ell)}, \tag{10}$$

where $s(k,\ell) = \left|C^k \cap \{\ell\}\right|/M$*,* $s(k) = \left|C^k\right|/M$ *with* C^k *being the set of meta-classes in* $\mathbf{c}$*, and* $s(\ell) = |\{\ell\}|/M = 1/M$ *that belong to meta-class* k*. Then,* $\max_{\mathbf{c}} I(\mathbf{y}, \mathbf{c})$ *is an increasing function of* q*.*

The training process of a q-ary ECNN is the same as that of a binary ECNN introduced in the previous sections except that during training the encoder of a q-ary ECNN outputs q bits, i.e., $\mathbf{z}_n \in \mathbb{R}^q, n = 0, \ldots, N-1$. To decode, we may convert the q-ary code matrix $\mathbf{M}$ into its binary version and then apply the decoding processing (same to the binary ECNN) to decode. More details about q-ary ECNN implementation and experiments can be found in the supplementary material.

Experiments

In this section, we evaluate the robustness of ECNN under the adversarial attacks with different attack parameters. The details of these adversarial attacks are provided in the supplementary material. We also discuss some recently developed ensemble methods in the supplementary material, among which we experimentally compare our proposed ECNN with 1) the ECOC-based DNN proposed in (Verma and Swami 2019) as it shares a similar concept as ours and 2) ADP-based ensemble method proposed in (Pang et al. 2019) as it is the only method among the aforementioned ones that trains the ensemble in an end-to-end fashion. Another reason for choosing these two methods as baseline benchmarks is their reported classification accuracies under adversarial attacks are generally better than the other ensemble methods. Due to the page limitation, more experimental results can be found in the supplementary material. [1]

Setup

We test on two standard datasets: MNIST (LeCun et al. 1998), CIFAR-10 and CIFAR-100 (Krizhevsky and Hinton 2009). For the encoder of ECNN, we constrain $h_n = h$ for $n = 0, \ldots, N-1$ and $g_{\theta_n} = g_n^{\{2\}} \circ g^{\{1\}}$ for $n = 0, \ldots, N-1$ so that each composite function becomes $h \circ g_n^{\{2\}} \circ g^{\{1\}}$. We use ResNet20 (He et al. 2016) to construct $h \circ g_n^{\{2\}} \circ g^{\{1\}}$. To recap, ResNet20 consists of three stacks of residual units where each stack contains three residual units, so there are nine residual units in ResNet20. With the intent of maintaining low computational complexity, we construct the shared feature extraction function $g^{\{1\}}$ using the first eight residual units in ResNet20 while leaving the last one to $g^{\{2\}}$. The shared prediction function h is a simple Dense layer. The detailed structure of ECNN is available in the supplementary material. In the following, we use ECNN_γ^N to denote an ECNN, trained using a trade-off parameter γ used in (7), with N binary classifiers. In all our result tables, bold indicates the best performer in a particular row.

Performance Under White-box Attacks

White-box adversaries have knowledge of the classifier models, including training data, model architectures and parameters. We test the performance of ECNN in defending against white-box attacks. The default parameters used for different attack methods are provided in the supplementary material. We compare ECNN with two state-of-the-art ensemble methods:

1. The adaptive diversity promoting (ADP) ensemble model, proposed by (Pang et al. 2019), for which we use the same architecture and model parameters as reported therein. Specifically, we use $\mathrm{ADP}_{2,0.5}$ with three ResNet20s being its meta classifiers. Note that the optimal solution of ADP is attained when $M-1$ is divisible by the number of base classifiers N. For $M = 10$, using $N = 3$ is optimal for ADP and increasing N beyond that does not improve its performance.

[1] Our experiments are run on a GeForce RTX 2080 Ti GPU.

Attack	Para.	$\mathrm{ADP}_{2,0.5}$	TanhEns16	$\mathrm{ECNN}_{0.1}^{30}$
None	-	**99.7**	99.5	99.4
PGD	$\epsilon = 0.3$	0.2	79.2	**88.4**
C&W	$\kappa = 1$	87.1	97.0	**99.4**
BSA	$\alpha = 0.8$	51.0	95.0	**99.3**
JSMA	$\gamma = 0.6$	1.6	84.2	**99.0**
# params	-	818,334	401,168	490,209

Table 3: Classification accuracy (%) on adversarial MNIST examples.

Attack	Para.	$\mathrm{ADP}_{2,0.5}$	TanhEns16	$\mathrm{ECNN}_{0.1}^{30}$
None	-	**93.4**	87.5	85.1
PGD	$\epsilon = 0.04$	2.5	63.1	**71.6**
C&W	$\kappa = 1$	4.4	68.0	**80.8**
BSA	$\alpha = 0.8$	4.3	61.3	**78.7**
JSMA	$\gamma = 0.2$	15.8	68.2	**84.2**
# params		819,198	2,313,104	490,497

Table 4: Classification accuracy (%) on adversarial CIFAR-10 examples.

Attacks	Para.	ResNet20	$\mathrm{ECNN}_{0.02}^{80}$
None	-	62.2	**61.3**
PGD	$\epsilon = 0.04$	5.3	**36.3**
BIM	$\epsilon = 0.04$	6.3	**42.9**
C&W	$\kappa = 1$	12.4	**52.0**

Table 5: Classification accuracy (%) on adversarial CIFAR-100 examples.

2. The ECOC-based neural network proposed by (Verma and Swami 2019). We choose the most robust model named TanhEns16 reported therein, which stands for an ensemble model where the $\tanh$ function is applied element-wise to the logits, and a Hadamard matrix of order 16 is used. As reported in (Verma and Swami 2019), the TanhEns16 splits the whole network into four independent subnets and each outputs 4-bit codeword.

When generating adversarial examples, as pointed out by (Tramèr et al. 2020) there are some precautions that should be taken: 1) at the decoder, we avoid taking the $\log$ of the logits before feeding them into the softmax function because taking $\log$ is numerically unstable, which leads to weak adversarial examples, and 2) we replace the softmax cross entropy loss function in PGD attack with the hinge loss proposed by (Carlini and Wagner 2017) in order to stabilize the process of crafting adversarial examples.

The classification results on MNIST are shown in Table 3. We can see that while maintaining the state-of-the-art accuracy on normal images, $\mathrm{ECNN}_{0.1}^{30}$ improves the adversarial robustness as compared to the other two methods. For the most effective attack in this experiment, i.e., PGD attack, ECNN shows a $88.4\% - 79.2\% = 9.2\%$ improvement over TanhEns16 and a $88.4\% - 0.2\% = 88.2\%$ improvement over $\mathrm{ADP}_{2,0.5}$. In terms of the number of trainable parameters, $\mathrm{ECNN}_{0.1}^{30}$ uses $\frac{490,209-401,168}{401,168} = 22.2\%$ more parameters than TanhEns16 and $\frac{818,334-490,209}{818,334} = 40.1\%$ less than $\mathrm{ADP}_{2,0.5}$.

For CIFAR-10, we see from Table 4 that $\mathrm{ADP}_{2,0.5}$ has the best classification accuracy on normal examples but it fails to make any reasonable predictions under adversarial attacks. This is mainly due to the use of strong attack parameter settings. ECNN is consistently more robust than the competitors under different adversarial attacks. In particular, $\mathrm{ECNN}_{0.1}^{30}$ achieves an absolute percentage point improvement over $\mathrm{ADP}_{2,0.5}$ of up to 76.4% (for C&W) and over TanhEns16 of 8.5% (for PGD) to 17.4% (for BSA) for different attacks. Moreover, the number of trainable parameters used in ECNN is $\frac{490,497}{819,198} = 59.88\%$ of that used in $\mathrm{ADP}_{2,0.5}$ and $\frac{490,497}{2,313,104} = 21.21\%$ of that used in TanhEns16.

For CIFAR-100, Table 5 shows that the most effective attack causes the classification accuracy to drop relatively by $40.7\% = \frac{61.3-36.3}{61.3}$ for $\mathrm{ECNN}_{0.02}^{80}$ and by $91.5\% = \frac{62.2-5.3}{62.2}$ for ResNet20. Due to limited computational resources, we restrict ourselves to using ResNet20 as the base classifiers in ECNN. We believe that replacing ResNet20 with a more sophisticated neural network will lead to better classification accuracy on both normal and adversarial examples.

ECNN is compatible with many defense methods such as adversarial training (AdvT).Experimental results with AdvT are given in Table 6, where AdvT augments the model's orignal loss (e.g., (7) for ECNN) caused by normal training examples with the loss caused by adversarial examples in each mini-batch. The ratio of adversarial examples and normal ones in each mini-batch is 1:1. In the training phase, we use PGD, where softmax cross-entropy loss is used, at $\epsilon = 0.04$ with 50 iterations to craft adversarial examples. In the testing phase, we run PGD at $\epsilon = 0.03$ for 200 iterations to attack. As can be observed, ECNN itself is superior to pure AdvT. ECNN+AdvT achieves the best performance.

Performance Under Black-box Physical World Attack

We conduct tests on the German Traffic Sign Recognition Benchmark (GTSRB) (Stallkamp et al. 2012) under black-box setting, where adversaries do not know the model internal architectures or training parameters. An adversary crafts adversarial examples based on a substitute model and then feed these examples to the original model to perform the attack. We train ResNet20 and ECNN on the GTSRB and test

Attacks	Para.	ResNet20	ResNet20 + AdvT
PGD	$\epsilon = 0.03$	10.9	51.9
		$\mathrm{ECNN}_{0.1}^{30}$	$\mathrm{ECNN}_{0.1}^{30}$+AdvT
		59.6	69.4

Table 6: Classification accuracy (%) on adversarial CIFAR-10 examples.

(a) Normal examples (b) Adversarial examples

Fig. 2: German traffic examples where the adversarial examples are crafted by (Sitawarin et al. 2018).

Attack	ResNet20	$\text{ECNN}^{30}_{0.2}$
None	95.5	**97.2**
OptProjTran	48.2	**79.6**

Table 7: Classification accuracy (%) on 392 adversarial GTSRB examples by (Sitawarin et al. 2018).

them on 12630 normal examples and 392 adversarial examples crafted by OptProjTran method [2] proposed in (Sitawarin et al. 2018) using a custom multi-scale CNN (Sermanet and LeCun 2011). The traffic sign examples are shown in Fig. 2a and Fig. 2b. The classification results are summarized in Table 7.

Impact of Ensemble Diversity

When removing the ensemble diversity from the optimization, the performance decreases significantly. Specifically, using the same shared front network for a fair comparison, we obtain 41.2% accuracy after applying PGD attack using $\text{ECNN}^{10}_{0.1}$ (cf. Table 2 of supplementary) on CIFAR-10, but we only achieve 17.0% accuracy using ECNN^{10}_{0}.

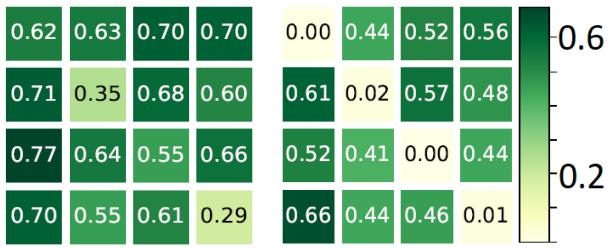
Transferability Study

Transferability study is carried out using $\text{ECNN}^{10}_{\gamma}$ on CIFAR-10, where the (i, j)-th entry shown in Fig. 3 is the classification accuracy using the i-th network as the substitute model to craft adversarial examples by running PGD at $\epsilon = 0.04$ for 200 iterations and feeding to the j-th network. It can be seen that 1) ECNN training yields low transferability among the binary classifiers and 2) having diversity control improves the robustness on individual networks.

Impact of Parameter Sharing

ECNN performs parameter sharing among binary classifiers at both ends of the encoder. Sharing $g^{\{1\}}$ is inspired by the lesson from transfer learning (Yosinski et al. 2014) that features are transferable between neural networks when they are performing similar tasks. Due to Lemma 2, sharing h is to avoid the features extracted from different binary classifiers to be permutations of each other for the same sample

[2]The adversarial examples generated by this attack are available at https://github.com/inspire-group/advml-traffic-sign.



(a) w/ ens. divers., $\gamma = 0.1$ (b) w/o ens. divers., $\gamma = 0$

Fig. 3: Adversarial transferability (binary classification accuracy) among binary classifiers in $\text{ECNN}^{10}_{\gamma}$ on CIFAR-10. The transferability among the first four networks is shown.

Attack	Para.	w/ para. share	w/o para. share
None	-	85.1	79.2
PGD	$\epsilon = 0.04$	71.6	51.1
C&W	$\kappa = 1$	80.8	73.2
# params	-	490,497	8,194,590

Table 8: Classification accuracy (%) on adversarial CIFAR-10 examples using $\text{ECNN}^{30}_{0.1}$ w/ and w/o parameter sharing.

x. Table 8 shows that performing parameter sharing yields better classification accuracy on normal and adversarial examples than the one with no parameter sharing, i.e., each binary classifier is a composite function $h_n \circ g_n^{\{2\}} \circ g_n^{\{1\}}$. This is mainly because the latter which contains too many parameters overfits the data.

Conclusion

In this paper, we have presented a robust neural network ECNN that is inspired by error-correcting codes and analyzed its properties and proposed a training method that trains the binary classifiers jointly. Designing a code matrix by optimizing both the Hamming distance between rows and VI distance between columns makes ECNN more robust against adversarial examples. We found that performing parameter sharing at two ends of the ECNN's encoder improves ensemble's robustness while significantly reducing the number of trainable parameters.

Acknowledgements

This research is supported in part by A*STAR under its RIE2020 Advanced Manufacturing and Engineering (AME) Industry Alignment Fund – Pre Positioning (IAF-PP) (Grant No. A19D6a0053) and Industry Alignment Fund (LOA Award I1901E0046). The computational work for this article was partially performed on resources of the National Supercomputing Centre, Singapore (https://www.nscc.sg).

References

Abbasi, M.; and Gagné, C. 2017. Robustness to Adversarial Examples through an Ensemble of Specialists. In *Proc. Int. Conf. Learning Representations Workshop*.

Andor, D.; Alberti, C.; Weiss, D.; Severyn, A.; Presta, A.; Ganchev, K.; Petrov, S.; and Collins, M. 2016. Globally Normalized Transition-Based Neural Networks. In *Proc. Annu. Meeting Assoc, Comput. Linguistics*.

Carlini, N.; and Wagner, D. 2017. Towards evaluating the robustness of neural networks. In *Proc. IEEE Symp. Security and Privacy*.

Dietterich, T. G.; and Bakiri, G. 1994. Solving multiclass learning problems via error correcting output codes. *J. Artificial Intell. Res.* 2(1): 263–286.

Gamal, A.; Hemachandra, L.; Shperling, I.; and Wei, V. 1987. Using simulated annealing to design good codes. *IEEE Trans. Inf. Theory* 33(1): 116–123.

García-Pedrajas, N.; and Fyfe, C. 2008. Evolving output codes for multiclass problems. *IEEE Trans. Evol. Comput.* 12(1): 93–106.

Goodfellow, I. J.; Shlens, J.; and Szegedy, C. 2015. Explaining and harnessing adversarial examples. In *Proc. Int. Conf. Learning Representations*.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proc. Conf. Comput. Vision Pattern Recognition*.

Hendrycks, D.; and Gimpel, K. 2017. A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks. In *Proc. Int. Conf. Learning Representations*.

Hinton, G.; Deng, L.; Yu, D.; Dahl, G. E.; Mohamed, A.; Jaitly, N.; Senior, A.; Vanhoucke, V.; Nguyen, P.; Sainath, T. N.; and Kingsbury, B. 2012. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Process. Mag.* 29(6): 82–97.

Kirkpatrick, S.; Gelatt, C. D.; and Vecchi, M. P. 1983. Optimization by simulated annealing. *Science* 220(4598): 671–680.

Krizhevsky, A.; and Hinton, G. 2009. Learning multiple layers of features from tiny images. *Master's thesis, Department of Computer Science, University of Toronto* .

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet classification with deep convolutional neural networks. In *Proc. Advances Neural Inf. Process. Syst.*

Kurakin, A.; Goodfellow, I. J.; and Bengio, S. 2017. Adversarial examples in the physical world. In *Proc. Int. Conf. Learning Representations*.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* 86(11): 2278–2324.

LeCun, Y.; Corte, C.; and Burges, C. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2. (Last accessed: Dec 1, 2020).

Meilă, M. 2003. Comparing clusterings by the variation of information. In *Proc. Annu Conf. Learning Theory*.

Meng, D.; and Chen, H. 2017. Magnet: a two-pronged defense against adversarial examples. In *Proc. SIGSAC Conf. Comput. Commun. Security*.

Mądry, A.; Makelov, A.; Schmidt, L.; Tsipras, D.; and Vladu, A. 2018. Towards deep learning models resistant to adversarial attacks. In *Proc. Int. Conf. Learning Representations*.

Pang, T.; Xu, K.; Du, C.; Chen, N.; and Zhu, J. 2019. Improving adversarial robustness via promoting ensemble diversity. In *Proc. Int. Conf. Mach. Learning*.

Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; and Swami, A. 2016a. The limitations of deep learning in adversarial settings. In *Proc. IEEE Eur. Symp. Security Privacy*.

Papernot, N.; McDaniel, P.; Wu, X.; Jha, S.; and Swami, A. 2016b. Distillation as a defense to adversarial perturbations against deep neural networks. In *Proc. IEEE Symp. Security and Privacy*.

Pujol, O.; Radeva, P.; and Vitria, J. 2006. Discriminant ECOC: A heuristic method for application dependent design of error correcting output codes. *IEEE Trans. Pattern Anal. Mach. Intell.* 28(6): 1007–1012.

Samangouei, P.; Kabkab, M.; and Chellappa, R. 2018. Defense-GAN: protecting classifiers against adversarial attacks using generative models. In *Proc. Int. Conf. Learning Representations*.

Sen, S.; Ravindran, B.; and Raghunathan, A. 2020. EMPIR: Ensembles of Mixed Precision Deep Networks for Increased Robustness Against Adversarial Attacks. In *Proc. Int. Conf. Learning Representations*.

Sermanet, P.; and LeCun, Y. 2011. Traffic sign recognition with multi-scale Convolutional Networks. In *Proc. Int. Joint Conf. Neural Networks*, 2809–2813.

Sitawarin, C.; Bhagoji, A.; Mosenia, A.; Mittal, P.; and Chiang, M. 2018. Rogue Signs: Deceiving Traffic Sign Recognition with Malicious Ads and Logos. In *Deep Learning and Security Workshop*.

Stallkamp, J.; Schlipsing, M.; Salmen, J.; and Igel, C. 2012. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks* 32(1): 323–332.

Szegedy, C.; Vanhoucke, V.; Ioffe, S.; Shlens, J.; and Wojna, Z. 2016. Rethinking the Inception Architecture for Computer Vision. In *Proc. Conf. Comput. Vision Pattern Recognition*.

Szegedy, C.; Zaremba, W.; Sutskever, I.; Bruna, J.; Erhan, D.; Goodfellow, I.; and Fergus, R. 2013. Intriguing properties of neural networks. In *Proc. Int. Conf. Learning Representations*.

Tramèr, F.; Carlini, N.; Brendel, W.; and Madry, A. 2020. On Adaptive Attacks to Adversarial Example Defenses. *ArXiv* abs/2002.08347.

Verma, G.; and Swami, A. 2019. Error correcting output codes improve probability estimation and adversarial robustness of deep neural networks. In *Proc. Advances Neural Inf. Process. Syst.*

Xu, W.; Evans, D.; and Qi, Y. 2018. Feature Squeezing: Detecting Adversarial Examples in Deep Neural Networks. In *Proc. Annu. Network and Distributed System Security Symp.*

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H. 2014. How transferable are features in deep neural networks? In *Proc. Advances Neural Inf. Process. Syst.*