Supplementary Materials for "Improving Robustness Against Stealthy Weight Bit-Flip Attacks by Output Code Matching"

Ozan Özdenizci[1,2] and Robert Legenstein[1]
[1] Institute of Theoretical Computer Science, Graz University of Technology, Graz, Austria
[2] TU Graz - SAL Dependable Embedded Systems Lab, Silicon Austria Labs, Graz, Austria
{ozan.ozdenizci,robert.legenstein}@igi.tugraz.at

1. Details on the Experimental Settings

In this section we describe further details on our model training and evaluation settings. Our implementations and saved model checkpoints are available at: https://github.com/IGITUGraz/OutputCodeMatching.

1.1. Datasets and Preprocessing

We used CIFAR-10, CIFAR-100 and ImageNet datasets in our experiments. CIFAR-10 and CIFAR-100 datasets [3] both consist of 50,000 training and 10,000 test images, each one belonging to one of 10 or 100 classes. ImageNet dataset [8] consists of 1,281,167 training images, and 50,000 validation images from 1000 classes. Note that we used the readily available validation set of the original ImageNet database as our test set. We follow the conventional data augmentation pipelines for both datasets during model training. Specifically for CIFAR-10 and CIFAR-100 we randomly cropped the images into 32×32 dimensions by padding zeros for at most 4 pixels around it, and randomly performed a horizontal flip. For ImageNet we randomly cropped the images and resized them back to 224×224 dimensions, also followed by a random horizontal flip. We normalized the image pixel values using the conventional metrics estimated on the training set of each dataset.

1.2. Optimization Configurations

We performed parameter optimization for all models using stochastic gradient descent (SGD) with momentum. Models were trained for 160 epochs with a batch size of 128 on CIFAR-10 and CIFAR-100, whereas a batch size of 256 was used for models on ImageNet. We used a piecewise constant decay learning rate with the initial learning rate set to 0.1 and divided by 10 during optimization when 50% and 75% of the total number of epochs are completed. A weight decay constant of 0.0005 for CIFAR-10 and CIFAR-100, and 0.0001 for ImageNet were used for all models that are trained to minimize softmax cross-entropy loss which yielded the best performance. For all output code matching (OCM) models which were trained end-to-end under the l_1-distance loss objective, a weight decay constant of 0.0001 was used as it resulted in higher clean test accuracies.

In CIFAR-10/100 experiments, OCM networks were trained end-to-end from scratch for 160 epochs. We performed OCM for ResNet-50 models trained on ImageNet by finetuning pre-trained networks. Specifically, we used the feature encoder weights of pre-trained vanilla models (trained for 100 epochs using the momentum SGD configuration described earlier) in an equivalent OCM network which had a different output dense layer with the new code length dimensionality. Output dense layer was hence redefined starting from random initialization, and all the weights until the output dense layer of the pre-trained network was used to initialize the feature encoder parameters. We then finetuned OCM models for a duration of 60 epochs using an Adam optimizer with an initial learning rate of 0.001, and the same piecewise constant decay schedule.

All code and software were implemented with the PyTorch 1.9.0 library [5], and experiments were performed using GPU hardware of types NVIDIA GeForce GTX 2080Ti, NVIDIA A40, NVIDIA Quadro P8000 and P6000.

1.3. Designing Bit String Output Codes

Our output code matching approach uses bit strings $\mathbf{S}_y \in \{-1,1\}^N$ of length N for each class $y \in \{1,\dots,C\}$, such that the network predicts this code instead of the usual one-hot encoded target vector. To reduce attack stealthiness we use output codes that are partially overlapping across classes. We used Hadamard matrices constructed via Sylvester's method to obtain matrices of order 2^k for our design of partially overlapping bit string codes [9]. Specifically, Sylvester's construction iteratively obtains $\mathbf{H}_{2^k}$ by:

$$\mathbf{H}_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}, \; \mathbf{H}_{2^k} = \begin{bmatrix} \mathbf{H}_{2^{k-1}} & \mathbf{H}_{2^{k-1}} \\ \mathbf{H}_{2^{k-1}} & -\mathbf{H}_{2^{k-1}} \end{bmatrix}, \quad (1)$$

which for any $\mathbf{H}_{2^k}$ then yields $\mathbf{H}_{2^k}\mathbf{H}_{2^k}^T = 2^k\mathbf{I}_{2^k}$. While the Hamming distance between the output representations

of any two class is 2 for conventional one-hot encoding, using Hadamard-type coding ensures that the Hamming distance between any two codes is $N/2$ [6, 9]. For classification problems with C-classses where C is not a power of 2, we randomly selected C rows of the $N \times N$ Hadamard matrix. For instance the following bit string codes were used for OCM_{16} ResNet-20 models trained on CIFAR-10:

$$
\begin{aligned}
\mathbf{S}_1 &= [\ 1\ 1\ 1\ 1\ -1\ -1\ -1\ -1\ 1\ 1\ 1\ 1\ -1\ -1\ -1\ -1\ ], \\
\mathbf{S}_2 &= [\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ -1\ ], \\
\mathbf{S}_3 &= [\ 1\ -1\ -1\ 1\ -1\ 1\ 1\ -1\ 1\ -1\ -1\ 1\ -1\ 1\ 1\ -1\ ], \\
\mathbf{S}_4 &= [\ 1\ -1\ -1\ 1\ 1\ -1\ -1\ 1\ 1\ -1\ -1\ 1\ 1\ -1\ -1\ 1\ ], \\
\mathbf{S}_5 &= [\ 1\ -1\ -1\ 1\ 1\ -1\ -1\ 1\ -1\ 1\ 1\ -1\ -1\ 1\ 1\ -1\ ], \\
\mathbf{S}_6 &= [\ 1\ -1\ 1\ -1\ -1\ 1\ -1\ 1\ 1\ -1\ 1\ -1\ -1\ 1\ -1\ 1\ ], \\
\mathbf{S}_7 &= [\ 1\ 1\ -1\ -1\ 1\ 1\ -1\ -1\ 1\ 1\ -1\ -1\ 1\ 1\ -1\ -1\ ], \\
\mathbf{S}_8 &= [\ 1\ 1\ -1\ -1\ -1\ -1\ 1\ 1\ -1\ -1\ 1\ 1\ 1\ 1\ -1\ -1\ ], \\
\mathbf{S}_9 &= [\ 1\ -1\ 1\ -1\ 1\ -1\ 1\ -1\ 1\ -1\ 1\ -1\ 1\ -1\ 1\ -1\ ], \\
\mathbf{S}_{10} &= [\ 1\ -1\ -1\ 1\ -1\ 1\ 1\ -1\ -1\ 1\ 1\ -1\ 1\ -1\ -1\ 1\ ].
\end{aligned}
$$

Here, each class code bit string corresponds to one row of the Hadamard matrix of order 16, i.e., $\mathbf{H}_{16}$. Note that the first index of all the codes are always 1 in this case, meaning that the outputs would in fact be effectively treated as 15-dimensional codes to be predicted as the output of the first neuron will not change for any class in the training set.

1.4. Attack Configurations

Stealthy T-BFA: Attacker requires a set of source class samples $\mathcal{D}_{\text{src}}$ and auxiliary samples $\mathcal{D}_{\text{aux}}$ from other classes to realize T-BFA. We determined the size of the sets $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{aux}}$ in accordance with the original implementation[1] in [7]. For a targeted attack from source class $s \rightarrow t$, we used half of all the samples from class s available in the the test split of the dataset. More specifically for CIFAR-10 we used 500 samples for $\mathcal{D}_{\text{src}}$ from class s, and another 500 samples for $\mathcal{D}_{\text{aux}}$ from any other class except s. In the same manner, for CIFAR-100 we used 50 samples and for ImageNet we used 25 samples for both $\mathcal{D}_{\text{src}}$ and $\mathcal{D}_{\text{aux}}$.

Using these samples we run stealthy T-BFA attacks until the attacker achieves 100% accuracy over all samples in $\mathcal{D}_{\text{src}}$ to be classified as t. We report attack success rate (ASR) as an attack generalization measure in our results, estimated as the accuracy of the held-out samples in the test set belonging to class s (e.g., for CIFAR-10 the unused 500 test samples belonging to class s) to be also classified as class t under the attacked model. We calculate post-attack accuracy (PA-ACC) for T-BFA on the test set except the samples from class s and used auxiliary samples in $\mathcal{D}_{\text{aux}}$.

For CIFAR-100 and ImageNet, only the first 50 classes were considered as a target or source class in the attacks. We used each of the 50 classes as the source class twice, with the attack aiming to misclassify the source class images as either the previous or next class label (i.e., $c_1 \rightarrow c_{50}$, $c_1 \rightarrow c_2$, $c_2 \rightarrow c_1$, $c_2 \rightarrow c_3$, ...). Each of these source-to-target settings were also repeated five times by changing the random choice of the auxiliary set $\mathcal{D}_{\text{aux}}$, and the samples drawn from the source class for $\mathcal{D}_{\text{src}}$. This yielded a set of 500 T-BFA experiments for CIFAR-100 and ImageNet. For CIFAR-10, we performed T-BFA similarly using all 10 classes as the source or target in 100 experiments (i.e., five repetitions of 20 source-to-target settings). Therefore all evaluation metrics are averaged over 500 experiments for CIFAR-100 and ImageNet, and 100 for CIFAR-10.

Stealthy TA-LBF: For each TA-LBF experiment the attacker requires a single source sample and auxiliary samples $\mathcal{D}_{\text{aux}}$ from the test set. We randomly draw 64 samples from the test set (belonging to any class) to construct an auxiliary set for each experiment. We evaluated TA-LBF on CIFAR-10 using 1000 single sample attacks[2] in total, where each one of the 10 classes is the target class for 100 different source images belonging to any other class. For stealthy TA-LBF we calculate ASR as the percentage of successful attacks across the 1000 attacks, and PA-ACC as the accuracy on the test set except the single attack source sample and the auxiliary samples used for attack optimization.

Section 3 details our formulation of the TA-LBF attack optimization objective, as the attack was trivially non-applicable to OCM networks due to its reliance to individual logits of the output softmax, which does not exist in OCM networks. We reported TA-LBF evaluations only for CIFAR-10 as our experiments with the adjusted TA-LBF attack revealed same evaluation results only for ResNet-20 models on CIFAR-10 as in [1]. We could not obtain successful TA-LBF attacks (i.e., ASR <5%) for the larger networks we used for CIFAR-100 and ImageNet.

2. Stealthy T-BFA Attacks on OCM

Stealthy T-BFA [7] attacks aim to misclassify the samples belonging to source class s as a target class t by solving the following optimization objective:

$$
\min_{\hat{\mathbf{B}}} \mathbb{E}_{\mathcal{D}_{\text{src}}} \left[\mathcal{L}(f(x; \hat{\mathbf{B}}); t) \right] + \mathbb{E}_{\mathcal{D}_{\text{aux}}} \left[\mathcal{L}(f(x; \hat{\mathbf{B}}); y) \right], \quad (2)
$$

with $f(x; \hat{\mathbf{B}})$ being the quantized DNN output and $\mathcal{L}$ the DNN training loss function, using a set of auxiliary samples $\mathcal{D}_{\text{aux}}$ and a set of source class samples $\mathcal{D}_{\text{src}}$. The solution is approximated by [7] using a heuristic progressive inter- and intra-layer bit search algorithm based on ranking the gradient of the training loss function.

For OCM we also used the DNN training loss $\mathcal{L}_{\text{OCM}}$ for the attacks as proposed in the original formulation [7], i.e.,

[1] https://github.com/adnansirajrakin/T-BFA

[2] https://github.com/jiawangbai/TA-LBF

l_1-norm of the distance between the target and the network output, yielding the stealthy T-BFA objective on OCM:

$$\min_{\hat{\mathbf{B}}} \mathbb{E}_{\mathcal{D}_{\text{src}}} \left[|f(x;\hat{\mathbf{B}}) - \mathbf{S}_t| \right] + \mathbb{E}_{\mathcal{D}_{\text{aux}}} \left[|f(x;\hat{\mathbf{B}}) - \mathbf{S}_y| \right], \quad (3)$$

where $\mathbf{S}_t$ is the output code for the target class, and $\mathbf{S}_y$ is the correct output code for x from class y that is sampled from $\mathcal{D}_{\text{aux}}$. For vanilla networks and models trained with piecewise clustering, we use the standard softmax cross-entropy loss for $\mathcal{L}(f(x;\hat{\mathbf{B}});y)$ in the attacks as in [7].

3. Stealthy TA-LBF Attacks on OCM

We denote the DNN output $f(x;\mathbf{B})$ as a composition of $g(x;\{\mathbf{B}^l\}_{l=1}^{L-1})$, the output classification layer with quantized weights $\mathbf{W}^L$, and output activation $\varphi(.)$. In a standard DNN, $\varphi(z)$ denotes a softmax activation with z being the pre-softmax logits. Stealthy TA-LBF [1] attacks aim to misclassify a single source sample as a target class t under the constraint $d_{\text{H}}(\mathbf{B},\hat{\mathbf{B}}) \leq k$, by solving the objective:

$$\min_{\hat{\mathbf{B}}^L} \mathcal{L}_{\text{eff}} + \gamma\, \mathbb{E}_{\mathcal{D}_{\text{aux}}} \left[\mathcal{L}(f(x;\hat{\mathbf{B}});y) \right], \quad (4)$$

where $\mathcal{L}$ is the training loss, k is the number of maximum bit-flips and γ is the trade-off parameter between stealthiness and the effectiveness loss $\mathcal{L}_{\text{eff}}$ given by:

$$\mathcal{L}_{\text{eff}} = \max(z_m - z_t + \eta, 0) + \max(z_s - z_m + \eta, 0), \quad (5)$$

with η a chosen margin parameter for pre-softmax logits, and $z_m = \max_{i\in\{0,\ldots,C\}\setminus\{s\}} z_i$, i.e., the second largest logit value for the source sample. The loss in Eq. (5) is minimized when both the pre-softmax target class logit z_t becomes greater than $z_m + \eta$, and the source class logit z_s becomes less than $z_m - \eta$. Thus, TA-LBF effectiveness significantly depends on changing the ranking of logits, as softmax assigns the highest probability to the largest logit.

One-hot encoding with independent logits does not exist in OCM. Therefore the original TA-LBF formulation is trivially non-applicable to our networks. To allow comparisons we re-formulate Eq. (4) under the same constraints as:

$$\min_{\hat{\mathbf{B}}^L} \mathcal{L}_{\text{bce}}(f(x;\hat{\mathbf{B}});t) + \gamma\, \mathbb{E}_{\mathcal{D}_{\text{aux}}} \left[\mathcal{L}_{\text{bce}}(f(x;\hat{\mathbf{B}});y) \right], \quad (6)$$

where $\mathcal{L}_{\text{bce}}$ computes the average of the binary cross-entropies across the N output units for OCM. To realize the binary cross-entropy computation, we interpret the OCM network output as one that has element-wise sigmoid activations where $\varphi(z) \in [0,1]$, and consider the bit string codes as $\mathbf{S}_y \in \{0,1\}^N$. We solve the constrained optimization objective from Eq. (6) using l_p-box alternating direction method of multipliers (ADMM) [11], in the same way as implemented by [1]. Note that for vanilla and piecewise clustering models, we also use the modified attack with Eq. (6), where $\mathcal{L}_{\text{bce}}$ is replaced by $\mathcal{L}_{\text{xent}}$, i.e., standard cross-entropy loss function. Our experiments with this adjusted attack revealed same CIFAR-10 evaluation results for the vanilla and piecewise clustering defended ResNet-20 models as reported in [1]. However we could not obtain successfully converging attacks for the larger networks we used for CIFAR-100 and ImageNet (which were not considered in [1]), despite our explorations with various parameter settings. We also did not obtain successful attacks on OCM when we considered the l_1-distance based $\mathcal{L}_{\text{OCM}}$ for Eq. (6).

We perform a similar parameter search as in [1] to find the optimal k and γ for each source sample to be attacked. We start from an initial value of $k = 5$ and $\gamma = 100$, and evaluate the attack until it is successful by multiplying γ with 0.5 at most 8 times. If the attack was not yet successful then we multiply k with 2 to similarly search again for γ. We perform the outer loop search for k 6 times. If the source sample is not classified as the target class t as a result of the manipulated $\hat{\mathbf{B}}$ for any k and γ combination, we consider the attack to be unsuccessful for this source sample.

Table 1. Stealthy T-BFA evaluations on ImageNet with ResNet-50 models (8-bit) trained with various regularization strength parameters λ for the piecewise clustering defense. Results in bold, yielding the highest # bit-flips, are presented in the main manuscript.

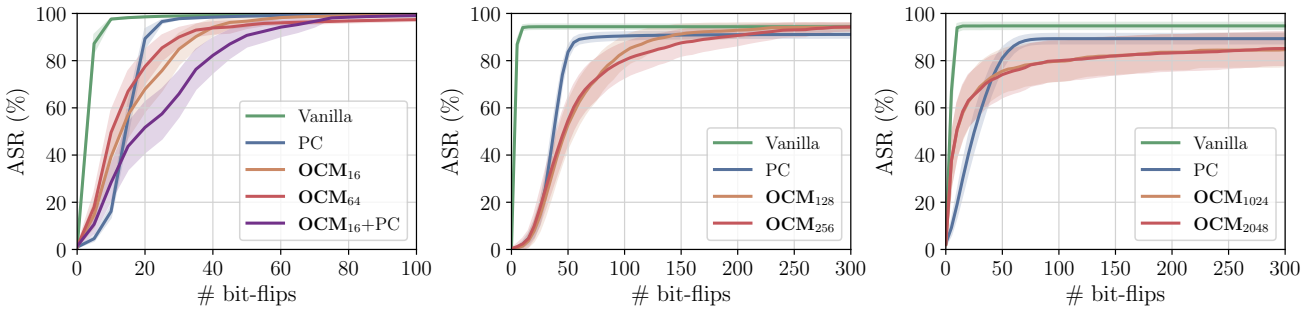
		Piecewise Clustering for ResNet-50		
		$\lambda = 0.001$	$\lambda = 0.0005$	$\lambda = 0.0001$
Training from scratch	Clean	63.28	**68.73**	74.64
	ASR	89.31	**89.32**	91.29
	PA-ACC	48.72	**54.81**	57.64
	# bit-flips	44.15	**48.65**	26.24
Finetuning pre-trained	Clean	73.74	74.61	75.90
	ASR	88.14	88.88	90.31
	PA-ACC	49.46	51.02	53.91
	# bit-flips	30.34	23.78	15.66

4. Piecewise Clustering Hyperparameters

We performed experiments with the piecewise clustering defense [2] in accordance with the public implementation [3]. Piecewise clustering trains quantized DNNs using a regularization term to enforce the quantized weights to have a bi-modal distribution, yielding the optimization problem:

$$\arg\min_{\{\mathbf{W}_l\}_{l=1}^L} \mathcal{L}_{\text{xent}} \left(f(x;\{\mathbf{W}_l\}_{l=1}^L);\, y \right) + \lambda \cdot \sum_{l=1}^{L} \left(||\, \mathbf{W}_l^+ - \mathbb{E}(\mathbf{W}_l^+)||_2 + ||\, \mathbf{W}_l^- - \mathbb{E}(\mathbf{W}_l^-)||_2 \right), \quad (7)$$

[3] https://github.com/elliothe/BFA



(a) CIFAR-10 with ResNet-20 (b) CIFAR-100 with WRN-28-4 (c) ImageNet with ResNet-50

Figure 1. Comparisons of stealthy T-BFA attack success rate (ASR) when the maximum allowed number of bit-flips by the attacker gradually increase. Models (8-bit) correspond to same networks evaluated in the main manuscript [4]. PC denotes the piecewise clustering defense. Shaded regions around the mean curve denotes ± 0.25 standard deviation.

where $\mathcal{L}_{\text{xent}}$ represents the standard cross-entropy loss function, λ is the regularization coefficient, $\mathbf{W}_l^+$ and $\mathbf{W}_l^-$ denote the positive and negative weight components of $\mathbf{W}_l$.

For CIFAR-10 and CIFAR-100 we used a piecewise clustering regularization constant of $\lambda = 0.001$. This λ was the common choice in [1, 2] for similar ResNet models, and we also did not obtain better models with different choices.

Nevertheless, we varied our choice of λ for the larger ResNet-50 models trained on ImageNet (both by end-to-end training from scratch, and by finetuning pre-trained networks) to examine model robustness, and obtained better models with $\lambda = 0.0005$. Table 1 presents our explorations of ResNet-50 models (8-bit) with piecewise clustering, where we explored same λ values as in [1, 2]. Our main observation was the large difference in robustness when we trained a ResNet-50 from scratch via piecewise clustering, as opposed to using a pre-trained network to be finetuned for a shorter amount of epochs. We performed finetuning of vanilla models (which were trained for 100 epochs) with piecewise clustering regularization for 60 epochs using an Adam optimizer with an initial learning rate of 0.0001, similar to our pipeline to obtain large-scale OCM models. Overall, as depicted in Tab. 1, faster training by finetuning did not yield significant robustness with piecewise clustering as opposed to the end-to-end regularized training from scratch (e.g., for $\lambda = 0.0005$ the # bit-flips was $2\times$ larger). We observed higher benign accuracies with finetuning that were close to the clean accuracy of the vanilla model, however with significantly less robustness in terms of the necessary number of bit-flips by a stealthy T-BFA attacker.

5. Constraining the Number of Bit-Flips

We investigate the stealthy bit-flip attack impact as the maximum number of allowed bit-flips gradually increase. Figure 1 denotes the increase in stealthy T-BFA ASR for all (8-bit) models trained on the three datasets. Note that the final evaluation metrics presented in the main manuscript [4] are the results obtained when the attacks are run until all source class set examples used by the stealthy T-BFA attacker are misclassified. Results show that our approach yields superior longitudinal resilience against the attacks as the number of bit-flips gradually increase. This robustness is even more pronounced when both defenses are combined (see OCM_{16}+PC in Fig. 1a). Figure 2 depicts an illustration of the early decrease in PA-ACC stealthiness with our results from the ResNet-20 models (8-bit) on CIFAR-10. Our results in the main manuscript [4] presented that PA-ACC was significantly lower for our models when attacks are run until they are successful on all $\mathcal{D}_{\text{src}}$ samples (PA-ACC for vanilla: 84.38%, PC: 76.78%, OCM_{16}: 53.22%, OCM_{64}: 46.39%, OCM_{16}+PC: 47.88%). Observations in Fig. 2 indicate that this decrease in PA-ACC occurs mainly within the early bit-flips, creating a large pre- and post-attack accuracy gap to break stealthiness even if the attacks are successful.

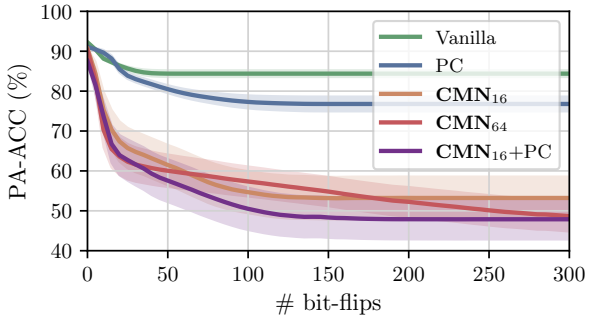


Figure 2. Comparisons of T-BFA attack stealthiness observed via PA-ACC, when the maximum allowed number of bit-flips by the attacker gradually increase. Results are shown for the ResNet-20 models (8-bit) on CIFAR-10 (shading: ± 0.25 standard deviation).

6. Additional Experiments

We explored the effectiveness of our approach with respect to using *mixup* [12] and *manifold mixup* [10] learning principles to regularize DNNs towards having decision

boundaries that transition linearly across classes and thus providing smoother uncertainty estimates. In principle, our approach of using overlapping output representations promotes a similar behavior by flattening learned representations across classes. However our defense is effective against stealthiness because the network is expected to predict target bit strings that are partially overlapping across classes, at test time as well. We performed experiments on CIFAR-10 with an 8-bit quantized ResNet-20, using mixup with $\alpha = 1$ [12], and manifold mixup with $\alpha = 2$ [10]. Required number of bit-flips (↑) for stealthy T-BFA were on average 28.9 and 52.3 respectively for mixup and manifold mixup, whereas it was 95.7 for OCM_{16} (and 281.8 for OCM_{64}). For these attacks, resulting PA-ACC (↓) values were 68.2% and 67.5% for the mixup methods, whereas it was 53.2% for OCM_{16}. Under stealthy TA-LBF number of bit-flips required for the attacker were on average 5.7 and 7.6 respectively for mixup and manifold mixup, whereas it was 31.1 for OCM_{16}. For these attacks the resulting PA-ACC values were 89.5%, 86.6% and 86.5% respectively. Thus, we demonstrated that the effectiveness of OCM mainly arises from using overlapping codes at test time, as sketched in Fig. 2 of the main manuscript [4].

References

[1] Jiawang Bai, Baoyuan Wu, Yong Zhang, Yiming Li, Zhifeng Li, and Shu-Tao Xia. Targeted attack against deep neural networks via flipping limited weight bits. In *International Conference on Learning Representations (ICLR)*, 2021. 2, 3, 4

[2] Zhezhi He, Adnan Siraj Rakin, Jingtao Li, Chaitali Chakrabarti, and Deliang Fan. Defending and harnessing the bit-flip based adversarial weight attack. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 14095–14103, 2020. 3, 4

[3] Alex Krizhevsky. Learning multiple layers of features from tiny images. *Technical Report, University of Toronto*, 2009. 1

[4] Ozan Özdenizci and Robert Legenstein. Improving robustness against stealthy weight bit-flip attacks by output code matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022. 4, 5

[5] Adam Paszke et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, 32:8026–8037, 2019. 1

[6] Morris Plotkin. Binary codes with specified minimum distance. *IRE Transactions on Information Theory*, 6(4):445–450, 1960. 2

[7] Adnan Siraj Rakin, Zhezhi He, Jingtao Li, Fan Yao, Chaitali Chakrabarti, and Deliang Fan. T-BFA: Targeted bit-flip adversarial weight attack. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. 2, 3

[8] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 1

[9] Jennifer Seberry and Mieko Yamada. Hadamard matrices, sequences, and block designs. *Contemporary Design Theory: A Collection of Surveys*, pages 431–560, 1992. 1, 2

[10] Vikas Verma, Alex Lamb, Christopher Beckham, Amir Najafi, Ioannis Mitliagkas, David Lopez-Paz, and Yoshua Bengio. Manifold mixup: Better representations by interpolating hidden states. In *International Conference on Machine Learning*, pages 6438–6447, 2019. 4, 5

[11] Baoyuan Wu and Bernard Ghanem. ℓ_p-Box ADMM: A versatile framework for integer programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(7):1695–1708, 2018. 3

[12] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *International Conference on Learning Representations (ICLR)*, 2018. 4, 5