Investigating Weight-Perturbed Deep Neural Networks With Application in Iris Presentation Attack Detection

Renu Sharma, Redwan Sony, Arun Ross
Michigan State University
{sharma90, sonymd, rossarun}@msu.edu

Abstract

Deep neural networks (DNNs) exhibit superior performance in various machine learning tasks, e.g., image classification, speech recognition, biometric recognition, object detection, etc. However, it is essential to analyze their sensitivity to parameter perturbations before deploying them in real-world applications. In this work, we assess the sensitivity of DNNs against perturbations to their weight and bias parameters. The sensitivity analysis involves three DNN architectures (VGG, ResNet, and DenseNet), three types of parameter perturbations (Gaussian noise, weight zeroing, and weight scaling), and two settings (entire network and layer-wise). We perform experiments in the context of iris presentation attack detection and evaluate on two publicly available datasets: LivDet-Iris-2017 and LivDet-Iris-2020. Based on the sensitivity analysis, we propose improved models simply by perturbing parameters of the network without undergoing training. We further combine these perturbed models at the score-level and at the parameter-level to improve the performance over the original model. The ensemble at the parameter-level shows an average improvement of 43.58% on the LivDet-Iris-2017 dataset and 9.25% on the LivDet-Iris-2020 dataset. The source code is available at https://github.com/redwankarimsony/WeightPerturbation-MSU.

1. Introduction

Deep Neural Networks (DNNs) have revolutionized the machine learning field through their superior performance in various tasks especially in the field of computer vision [12, 13, 22], natural language processing [6], and speech technology [5]. In essence, a DNN comprises a sequence of layers containing trainable parameters (weights and bias) to learn a complex mapping between input signals and output labels. For deploying DNNs in real-world applications, it is crucial to analyze their robustness or sensitivity to hardware/sensor noise introduction [2], environment changes [26] and adversarial attacks [9]. Sensitivity analysis also helps in building a quantized-weights model with commensurate performance [11, 28].

In the literature, sensitivity analysis of DNNs has been performed by perturbing either the input signal or the architectural parameters. The work in [8, 10, 14, 16, 17, 24] analyze DNN robustness by manipulating the input signals, whereas the work in [11, 21, 26, 28, 29] perturb architectural parameters to analyze robustness. Yeung *et al.* [31] provide a detailed sensitivity analysis of neural networks over input and parameter perturbations. In this work, we focus on the sensitivity analysis of DNNs when architectural parameters (learned weights) are perturbed.

The authors in [21, 26, 28, 29] provide a theoretical sensitivity analysis based on parameter perturbations. Shu and Zhu [21] propose an influence measure motivated by information geometry to quantify the effects of various perturbations to input signals and network parameters on DNN classifiers. Xiang *et al.* [29] design an iterative algorithm to compute the sensitivity of a DNN layer by layer, where sensitivity is defined as “the mathematical expectation of absolute output variation due to weight perturbation with respect to all possible inputs” [29]. Tsai *et al.* [26] study the robustness of the pairwise class margin function against weight perturbations. Weng *et al.* [28] compute a certified robustness bound for weight perturbations, within which a neural network will not make erroneous outputs. In addition, they also identify a useful connection between the developed certification and the challenge of weight quantization.

In this work, we empirically analyze the sensitivity of DNNs by manipulating their architectural parameters. We examine sensitivity of three widely used architectures (VGG [22], ResNet [12], and DenseNet [13]) under three types of parameter perturbations (Gaussian noise, weight zeroing and weight scaling). We apply the perturbations in two settings: over all the layers of a network simultaneously and over each layer at a time. Our work is motivated from [2], where they also empirically analyze the sensitivity of the pre-trained AlexNet and VGG16 networks to internal architecture and weight perturbations. However, our work is vastly different. Being motivated by their analysis, not only do we analyze the robustness or sensitivity of the newer networks, we also improve those models with different perturbation methods without any training. First, we extend the work by evaluating the sensitivity of heavily used CNN architectures in biometric tasks: VGG, ResNet, and DenseNet. Second, we perform additional weight manipulations (weight scaling, variants of weight zeroing, and additional setting of applying perturbations over the entire network parameters) in the sensitivity analysis. Third, we leverage the findings from the sensitivity analysis and propose an ensemble of perturbed models to improve the performance without any further training. Our main contributions are as follows:

1. We perform sensitivity analysis of three DNN architectures (VGG [22], ResNet [12] and DenseNet [13]) against parameter perturbations.
2. We apply a number of parameter perturbations (three types of perturbations and its variant in two settings) to analyze the sensitivity of deep neural networks in the context of iris presentation attack detection.
3. We leverage the sensitivity analysis to propose a better performing model by ensembling the perturbed models at two different levels: score-level and parameter-level.
4. We perform experiments using five datasets. Three of the datasets (IARPA, NDCLD-2015, Warsaw Postmortem v3) are used for training, whereas the others (LivDet-Iris-2017 and LivDet-Iris-2020) are used for testing. This represents a cross-dataset scenario, where training and testing are performed on different datasets.

The rest of the paper is organized as follows: Section 2 provides the details of various parameter perturbations used for the sensitivity analysis of DNNs; Section 3 describes the application scenario considered in this work; Section 4 explains the dataset and experimental setup; Section 5 provides the sensitivity analysis of the three architectures against the considered parameter perturbations; and Section 6 describes how we leverage the sensitivity analysis to generate an ensemble of perturbed models for improving performance. Finally, Section 7 summarizes the paper and provides future directions.

2. Parameter Perturbations

We explore the sensitivity of neural networks by perturbing their architectural parameters (weights and bias). From here on, we use the terms 'architectural parameters', 'parameters', and 'weights' interchangeably. To measure the sensitivity, we consider the change in the performance of the DNN when weights are perturbed. Let n input samples be $\{x_1, x_2, ..., x_n\}$ and their output be $\{y_1, y_2, ..., y_n\}$. Here, we labeled the positive class as '1' and the negative class as '0'. The predicted output values from a DNN approximator are $\{f(x_1, W_{org}), f(x_2, W_{org}), ..., f(x_n, W_{org})\}$, where W_{org} are the learned parameters. We measure the performance of the DNN in terms of True Detection Rate (TDR). TDR is a percentage of positive samples correctly classified:

$$TDR_{org} = \frac{\sum_i^n (f(x_i, W_{org}) > T)}{\sum_i^n y_i} * 100 \tag{1}$$

where, T is the threshold. The input sample with a predicted value above the threshold is considered a positive class. After weight perturbation, we estimate the output as $\{f(x_1, W_{mod}), f(x_2, W_{mod}), ..., f(x_n, W_{mod})\}$, where W_{mod} are the perturbed parameters. We then use these predicted values to measure the performance of DNN (TDR_{mod}). The higher the change in the performance ($|TDR_{org} - TDR_{mod}|$), the higher the sensitivity of the neural network to the particular perturbation.

We perturb the parameters in two settings: manipulating parameters of all layers simultaneously and manipulating parameters one layer at a time. The first setting aims to understand the overall sensitivity of DNNs, whereas the second setting examines which layer has more impact on the model. The higher the sensitivity, the lower the generalization of the DNN [17, 26]. The three perturbations we consider are Gaussian noise manipulation, weight zeroing, and weight scaling. These perturbations resemble (a) noise introduction due to defects in hardware implementations of neural networks [15], and (b) adversarial weight perturbations [9, 19] on open-sourced models. Eventually, the choice of perturbations is based on their simplicity. This work has also the potential of obtaining quantized or compressed DNN models, which consume less memory with equivalent performance. Details of these perturbations are as follows:

1. Gaussian Noise Manipulation: Here, we manipulate the original parameters of the layers by adding Gaussian noise sampled from a normal distribution of zero mean and scaled standard deviation. We control the scaling of the standard deviation by the scalar factor $\boldsymbol{\alpha}$. The modified parameters are defined as

$$W_{mod} = W_{org} + N(0, \boldsymbol{\alpha} * \sigma(W_{org})). \tag{2}$$

Here, W_{org} are the original parameters, W_{mod} are the modified parameters, and $N(\mu, \sigma)$ is the normal distribution. We calculate $\sigma(W_{org})$ for a particular layer by first flattening the parameter tensor to a 1-D array and then computing the standard deviation. So, the standard deviation and the Gaussian noise distribution will differ for each layer since $\sigma(W_{org})$ varies from layer to layer. Consequently, the *absolute* perturbations differ for each layer. However, *relative* perturbations are the same across layers.

2. Weight Zeroing: In the second manipulation, we randomly select a certain proportion of parameters and set them to zero. The portion of parameters is determined by a scalar factor $\boldsymbol{\beta}$. The modified parameters are represented as

$$W_{mod}[random(\boldsymbol{\beta}, W_{org})] = 0. \quad (3)$$

Here, $random(.,.)$ is the function that returns the index of $\boldsymbol{\beta}$ proportion of randomly selected parameters from the original set of parameters. We also define another version of weight zeroing, where weights are first sorted, and then $\boldsymbol{\beta}$ proportion of low-magnitude weights is set to zero.

3. Weight Scaling: The third perturbation scales the original parameters by a scalar factor $\boldsymbol{\gamma}$ as

$$W_{mod} = \boldsymbol{\gamma} * W_{org}. \quad (4)$$

3. Application Scenario

We perform sensitivity analysis in the context of iris presentation attack detection (PAD). A presentation attack (PA) occurs when an adversary presents a fake or altered biometric sample such as printed eyes, plastic eyes, or cosmetic contact lenses to circumvent the iris recognition system [1]. Our application is to detect these PAs launched against an iris system. We formulate the detection problem as a two-class problem based on DNNs, where the input is a near-infrared iris image and the output is a PA score (range from 0-1) which is based on a specified threshold labeled as "bonafide" or "PA".

4. Datasets and Experimental Setup

The training data we use to build our iris PAD models are IARPA, NDCLD-2015 [27] and Warsaw PostMortem v3 [25] datasets. The IARPA dataset is a proprietary dataset consisting of 19,453 bonafide irides and 4,047 presentation attack (PA) samples. From the NDCLD-2015 dataset, we use 2,236 cosmetic contact lens images for training. From the Warsaw PostMortem v3 dataset, 1,200 cadaver iris images from the first 37 cadavers are used for training. Testing is performed on the LivDet-Iris-2017 [30] and LivDet-Iris-2020 [3] datasets. Both of these are publicly available competition datasets for evaluating iris presentation attack detection performance. The LivDet-Iris-2017 dataset [30] consists of four subsets: Clarkson, Warsaw, Notre Dame, and IIITD-WVU. All subsets contain train and test partitions, and we use only the test partition. Warsaw and Notre Dame subsets further contains two splits in the test partition: 'Known' and 'Unknown'. The 'Known' split corresponds to the scenario where PAs of the same type or images from similar sensors are present in both train and test partitions, while the 'Unknown' split contains different types of PAs or images from different types of sensors in the train and test partitions. In our case, both test splits are considered as 'Unknown' type as we use different datasets for training. Such a testing scenario is referred to cross-dataset. However, we keep the original terminologies ('Known' and 'Unknown') of test splits in the work. The LivDet-Iris-2020 [3] consists of a single test split, and this scenario also corresponds to cross-dataset. Table 1 describes all training and test sets, along with the types of PAs and images present in them. In aggregate, both datasets provide a diverse set of PAs.

We use three iris PA detectors for sensitivity analysis. Two of the detectors utilize VGG19 [22] and ResNet101 [12] networks as their backbone architecture. The third detector is D-NetPAD [20], where the backbone architecture is DenseNet161 [13]. The D-NetPAD shows state-of-the-art performance on both LivDet-Iris-2017 and LivDet-Iris-2020 iris PAD competitions [3, 20]. Since D-NetPAD already had the state-of-the-art performance on the evaluation datasets and Smith *et. al.* [23] found that convolution-based networks can perform same as vision transformer at scale, we did not perform similar analysis or experiments on transformer-based models like ViT [7]. The convolutional networks we use require a cropped iris region resized to 224 × 224 as input. For training, we initialize the model with the weights from the ImageNet dataset [4] and then fine-tune the models using the training datasets described above. The learning rate was set to 0.005, the batch size was 20, the number of epochs was 50, the optimization algorithm was stochastic gradient descent with a momentum of 0.9, and the loss function used was cross-entropy.

We measure the sensitivity of these DNNs by evaluating their performance as a function of the weight perturbations. The performance is estimated in terms of TDR (%) at 0.2% False Detection Rate (FDR).[1] FDR is the percentage of bonafide samples incorrectly classified as PAs.[2] In Table 3, the row corresponding to the 'Original' method reports the performance of these models on the LivDet-Iris-2017 and LivDet-Iris-2020 datasets *before* weights were perturbed. On the LivDet-Iris-2017 dataset, ResNet101 performs the best (average 74.55% TDR), whereas on the LivDet-Iris-2020 dataset, D-NetPAD performs the best (90.22% TDR). We also provide information about the number of weights and bias parameters present in all three models (Table 2). The VGG19 architecture has the highest number of parameters, followed by the ResNet101 architecture.

5. Sensitivity Analysis

5.1. Gaussian Noise Addition

The Gaussian noise manipulation involves the addition of Gaussian noise to the original parameters. Figure 1a

[1] The threshold at this specific FDR was selected by the sponsor.

[2] ISO/IEC 30107-3:2023 specifies Attack Presentation Classification Error Rate (APCER) and Bonafide Presentation Classification Error Rate (BPCER) as evaluation metrics for PAD. TDR is 1−APCER, and FDR is the same as BPCER.

Table 1. Summary of training and test datasets along with the number of bonafide and PA iris images present in the datasets. The information about the sensors used to capture images is also provided. Here, “K. Test” means a known test set of the dataset, and “U. Test” means an unknown test set (see text for explanation).

Train/Test	**Train**			**Test**						
Datasets	**IARPA**	**NDCLD -2015**	**Warsaw PostMortem v3**	**LivDet-Iris-2017**						**LivDet-Iris-2020**
Dataset Subsets				**Clarkson (Cross-PA)**	**Warsaw (Cross-sensor)**		**Notre Dame (Cross-PA)**		**IIITD-WVU (Cross-Dataset)**	
Splits				**Test**	**K. Test**	**U. Test**	**K. Test**	**U. Test**	**Test**	
Bonafide	19,453	-	-	1,485	974	2,350	900	900	702	5,331
Print	1,005	-	-	908	2,016	2,160	-	-	2,806	1,049
Cosmetic Contacts	1,187	2,236	-	765	-	-	900	900	701	4,336
Artificial Eyes	1,804	-	-	-	-	-	-	-	-	541
Electronic Display	51	-	-	-	-	-	-	-	-	81
Cadaver Eyes	-	-	1,200	-	-	-	-	-	-	1,094
Sensor	COTS Iris Sensors x3[1]	IrisGuard AD100, IrisAccess LG4000	IriShield MK2120U	IrisAccess EOU2200	IrisGuard AD100	Aritech ARX-3M3C, Fujinon DV10X7.5A, DV10X7.5A-SA2 lens B+W 092 NIR filter	IrisGuard AD100, IrisAccess LG4000		IriShield MK2120U	Iris ID iCAM7000, IrisGuardAD100, IrisAccess LG4000, IriTech IriShield

[1] Specific sensor names withheld at sponsor’s request

Table 2. The number of parameters (weights and bias) present in all convolutional layers and the entire network of the VGG19, ResNet101, and D-NetPAD architectures.

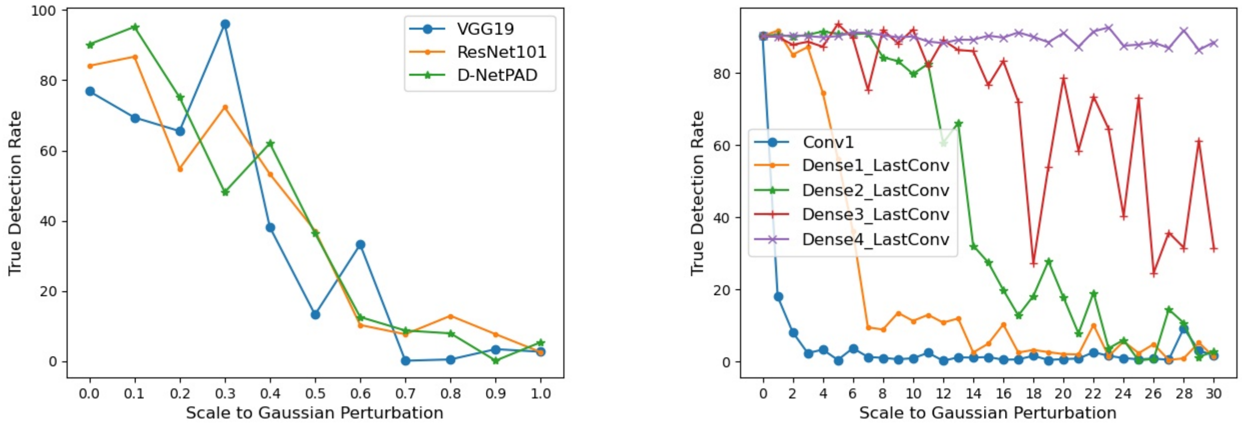
Architecture	**VGG19**	**ResNet101**	**D-NetPAD**
Weights	139,570,240	42,451,584	26,366,448
Bias	19,202	52,674	109,970
Total	139,589,442	42,504,258	26,476,418

shows the performance of all the networks when we perturb parameters of all layers with the Gaussian noise. The scale factor (α) used to modify the standard deviation is shown on the x-axis. Every data point in the figure represents a single performance of the model. From a trend standpoint, the performance of all networks decreases with an increase in the standard deviation. However, this decrease is not linear. In fact, there are some performance gains at certain scales. These scales are different for different networks. For instance, the VGG19 network shows improvement for α = 0.3, 0.6, and 0.9, ResNet101 for α = 0.1, 0.3, and 0.9, and D-NetPAD for α = 0.1, 0.4 and 1.0. Surprisingly, certain scales give higher performance than the original model, such as 0.1 scale for the ResNet101 and D-NetPAD models, and 0.3 scale for the VGG19 model. The results indicate that all three networks are sensitive to Gaussian noise perturbations when perturbations are applied over all layers of the network, and we cannot conclude which network is comparatively stable under these weight perturbations.

We further analyze the impact of perturbation at different layers on the performance of the models. We manipulate the parameters one layer at a time and observe the performance change. For the layer-wise analysis, we show the results for only the D-NetPAD model since the other two models also show similar performance trends. In the case of D-NetPAD, we select the first convolution layer and the last convolution layers of four dense blocks for perturbation. Figure 1b shows the performance of D-NetPAD when the individual layer’s parameters are perturbed. We observe that the initial layers have more influence on the performance of the D-NetPAD compared to the later layers. The model is highly robust to the perturbations in the last convolution layer of the fourth dense block, even at a scale factor of 30. Cheney *et al.* [2] also observe the higher impact of perturbations in the initial layers on the performance. Generally, initial layers focus more on capturing discriminative or representative features, whereas later layers are more responsible for generating decision boundaries. Manipulations to extracted features have more impact on the performance compared to a slight change in decision boundaries. Moreover, manipulation in initial layers changes feature maps of all subsequent layers and, hence, causes propagation of error. Change in middle layers exhibit large fluctuations in performance compared to the initial and later layers.

5.2. Weight Zeroing

The weight zeroing manipulation involves random selection of a particular fraction of weight parameters and setting them to zero. Figure 2a shows the performance of all three architectures when we manipulate the entire set of network parameters, while Figure 2b shows the performance of D-NetPAD when we perturb individual layers. Similar conclusions can be drawn from Figure 2a as drawn from Figure 1a that the overall performance of all three architectures decreases with an increase in the proportion of weights set to zero. However, certain perturbations give improved performance. For example zeroing 3% of weights improves the VGG19 network performance from 76.87% TDR (original) to 92.70% TDR. In the case of ResNet101, zeroing 3% of weights improves performance from 84.11% TDR (origi-



(a)

(b)

Figure 1. Gaussian noise manipulation: (a) Performance (TDR at 0.2% FDR) of VGG19, ResNet101, and D-NetPAD when weights and bias parameters of the entire network are perturbed. (b) Performance of D-NetPAD when the individual layer's parameters (weights and bias) are perturbed. Here, Conv1 means the first convolution layer of the D-NetPAD, Dense1_LastConv means the last convolution layer of the first dense block, and so on.

nal) to 88.88% TDR. Again, all three networks are sensitive to the zeroing out of randomly selected weights.

In the layer-wise setup (Figure 2b), the performance of D-NetPAD is stable except for the first convolution layer. This is due to the fact that the original weights of the convolution layers have a zero mean and a small standard deviation ranging from 0.10 (first convolution layer) to 0.01 (last convolution layer) as shown in Figure 3. Initial layers have a higher standard deviation compared to later layers, which makes the network more sensitive to the manipulations in the initial layers. A similar performance trend is observed in the VGG19 and ResNet101 networks as well.

Since most of the original weights are already close to 0, we apply a variant of weight zeroing where only low-magnitude weights are set to zero. Figure 4a shows the performance of all architectures when we manipulate the entire network in this fashion, while Figure 4b shows the performance of D-NetPAD on layer-wise manipulation. ResNet101 and D-NetPAD networks are observed to be robust to this manipulation as zeroing out even 33% of all weights does not affect their performance. VGG19 also shows robustness with only a 6% drop in performance, though its performance is not as stable as the ResNet101 and D-NetPAD networks. Figure 4b shows the sensitivity of the D-NetPAD on layer-wise perturbations. Zeroing out even 30% of the first convolution layer weights does not impact its performance. Remarkably, the manipulation in the last convolution layer of the first and second dense blocks shows a linear increase in performance. The performance of D-NetPAD increases from 90.22% TDR to 96.28% TDR upon manipulating the last convolution layer of the first dense block. This implies that we could zero out low-magnitude weights and reduce the size of the model without affecting its performance. This finding is useful in building a compressed DNN model with better time and memory efficiency to deploy on mobile or embedded devices.

5.3. Weight Scaling

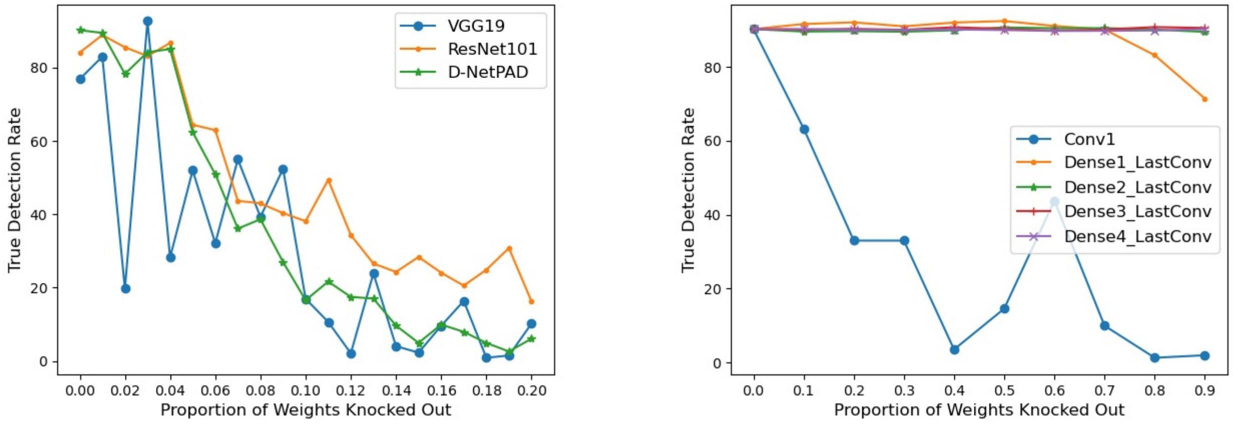
This manipulation scales the original parameters with a scalar value. Figure 5a shows the performance of all three architectures when we manipulate the entire set of network parameters, while Figure 5b presents the performance of D-NetPAD when we perturb specific layers. The performance at scale 1 indicates the original performance without weight perturbations. Weight perturbations across the entire network resulted in a radical drop in performance even with a small scalar factor (0.8 or 1.1). In the layer-wise manipulation, the initial layers show a higher impact on the performance of D-NetPAD compared to the later layers. The manipulation in the last convolution layer does not impact the performance even at a scaling factor of 10. A similar performance trend is observed on the VGG19 and ResNet101 networks as well.

5.4. Findings

Here are the main findings from the aforementioned analysis:

1. All three networks decrease in performance when perturbations are applied over the entire network. [3] However,

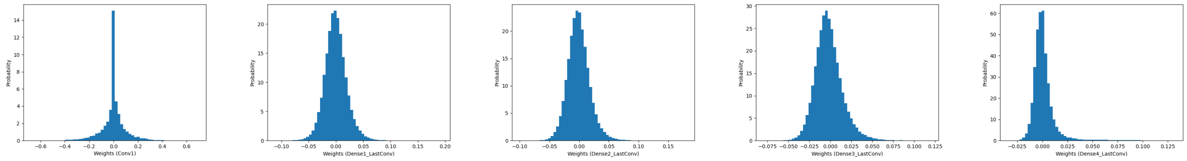
[3] The notion of 'significant' change in performance is a subject of future work.



(a)

(b)

Figure 2. Weight zeroing manipulation: (a) Performance (TDR at 0.2% FDR) of VGG19, ResNet101, and D-NetPAD when parameters of the entire network are perturbed. (b) Performance of D-NetPAD when the individual layer's parameters are perturbed.



(a) Conv1
μ: 0.0, σ: 0.103

(b) Dense1_LastConv
μ: 0.0, σ: 0.019

(c) Dense2_LastConv
μ: 0.0, σ: 0.018

(d) Dense3_LastConv
μ: 0.0, σ: 0.016

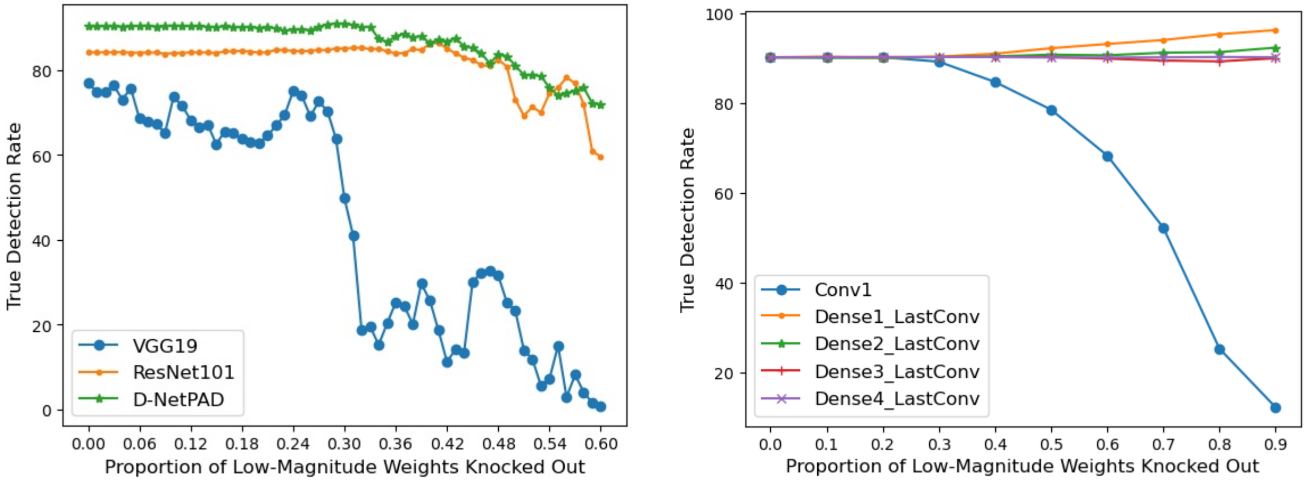
(e) Dense4_LastConv
μ: -0.0, σ: 0.012

Figure 3. Weight distribution of different layers of the trained D-NetPAD architecture. Mean (μ) and standard deviation (σ) are provided below each distribution.

the networks show robustness when low-magnitude weights are set to zero. The scaling of weights has a major negative impact on the performance of networks.
2. Layer-wise sensitivity analysis shows that perturbations in initial layers impacted the performance to a greater extent compared to the later layers. The weight distribution of all layers are zero-centered and later layers have a lower standard deviation compared to initial layers (Figure 3), making later layers less sensitive to weight zeroing and scaling perturbations as majority of their weights are already close to the zero mean. The zero-centered nature of weight distributions is also a reason why Gaussian noise perturbations have the most negative impact on the performance compared to the other perturbations.
3. Certain perturbations improve the performance of network models over the original one in both settings (entire network and layer-wise). This observation indicates that the parameters learned by the models during training are not optimum. Random change in the weights in their close vicinity shows improvement in the performance. Hence, there is further scope for optimizing weights.
4. Zeroing out low-magnitude weights results in better performance as well as reduces the size of the model.

6. Performance Improvement

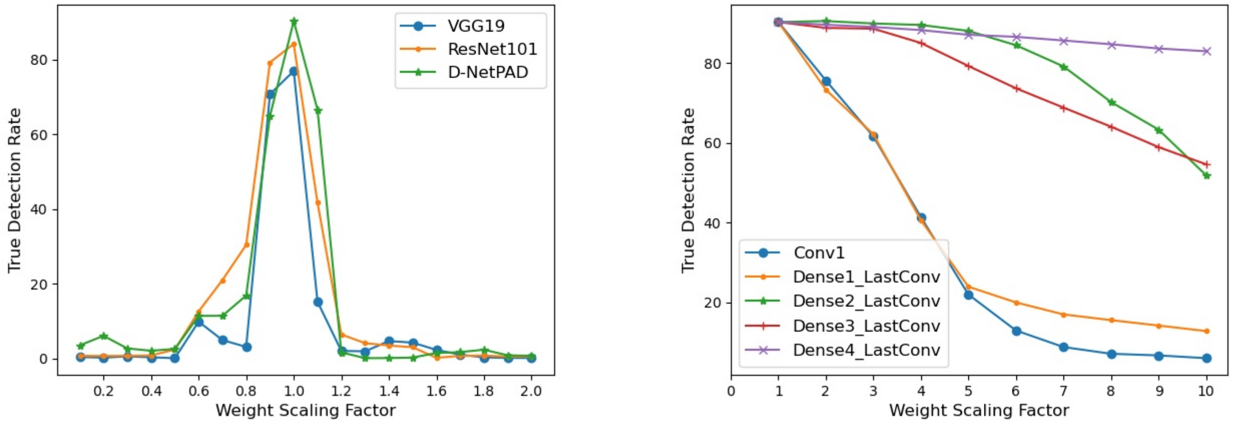
We observe that certain perturbations result in better performance, even higher than that of the original model. We leverage this observation and obtain better performing models using these perturbations without any additional training. In this regard, we explore two directions: the first is to find a single perturbed model which achieves good performance consistently, and the second is to create an ensemble of high-performing perturbed models. In the earlier part of the work, we analyzed the sensitivity of different architectures based on their performance on the LivDet-Iris-2020 dataset. Here, we select a high-performing perturbed model and validate its performance on the LivDet-Iris-2017 dataset. For the ensemble of models, we further explore two sub-directions based on the level of fusion. In the first, we simply fuse their decision scores using the sum rule. This level of fusion better spans the decision space and generalizes well to the test data [18]. However, it increases the inference time as decision scores are required from all the



(a)

(b)

Figure 4. Variant of the weight zeroing manipulation (low-magnitude weights are set to zero): (a) Performance (TDR at 0.2% FDR) of VGG19, ResNet101, and D-NetPAD when parameters of the entire network are perturbed. (b) Performance of D-NetPAD when individual layer's parameters are perturbed.



(a)

(b)

Figure 5. Weight scaling manipulation: (a) Performance (TDR at 0.2% FDR) of VGG19, ResNet101, and D-NetPAD when parameters of the entire network are perturbed simultaneously. (b) Performance of D-NetPAD when the individual layer's parameters are perturbed.

component models. The second level of fusion is at the model parameter-level, where we fuse the parameters by averaging and merge the component models into one model. This level of fusion better spans the parameter space and reduces the inference time as a decision is required from only one model. Details of these models are given below:

1. Original Model: The model utilizes originally trained parameters without any perturbation of the parameters.

2. Perturbed Model: In the case of VGG19, we create a perturbed model by setting 95% of the low-magnitude weights of the seventh convolution layer to zero. For the ResNet101 model, a perturbed model is formed by setting 40% of low-magnitude weights of the first convolution layer to zero, while for the D-NetPAD, 92% of the low-magnitude weights of the last convolution layer of the first dense block are set to zero. The selection of these perturbed models are based on their consistent high performance on the LivDet-Iris-2020 dataset (4b). We repeat the experiment 100 times for each of the high-performing models and select the one with the most consistent performance.

3. Ensemble Models at the Score-Level: We combine two consistent high-performing perturbed models by fusing their PA scores using the sum rule. For all three architectures, we fuse the above specified perturbed models with

Table 3. The performance of VGG19, ResNet101, and D-NetPAD models in terms of True Detection Rate (%, higher the better) at 0.2% False Detection Rate on the LivDet-Iris-2017 and LivDet-Iris-2020 datasets. The performance is shown on original model (no parameter perturbations), perturbed model and an ensemble of model.

Datasets	**LivDet-Iris-2017**						**LivDet-Iris-2020**
Subsets	**Clarkson**	**Warsaw**		**Notre Dame**		**IIITD-WVU**	
Splits	**Test**	**K. Test**	**U. Test**	**K. Test**	**U. Test**	**Test**	
VGG19 Model							
Original	51.32	86.25	10.12	**100**	**99.00**	1.44	76.87
Perturbed	54.88	91.12	9.08	**100**	97.78	1.58	**90.31**
Ensemble (Score-level)	66.17	**92.95**	7.23	**100**	98.00	3.14	89.53
Ensemble (Parameter-level)	**73.01**	84.92	**13.90**	99.78	97.78	**9.43**	88.26
ResNet101 Model							
Original	15.82	89.93	91.67	**100**	**99.44**	50.47	84.11
Perturbed	**23.01**	**95.33**	**94.65**	**100**	95.67	**58.14**	86.40
Ensemble (Score-level)	21.61	92.95	94.60	**100**	89.88	58.02	**91.07**
Ensemble (Parameter-level)	19.10	**95.33**	94.37	**100**	95.67	**58.14**	89.92
D-NetPAD Model							
Original	60.04	76.68	35.76	**100**	**99.33**	32.01	90.22
Perturbed	**68.54**	**94.94**	**53.02**	**100**	99.00	**50.35**	**96.86**
Ensemble (Score-level)	68.34	93.84	46.40	**100**	97.66	48.08	96.71
Ensemble (Parameter-level)	64.29	**94.94**	**53.02**	**100**	99.00	42.59	95.66

the model formed by adding Gaussian noise with $\alpha = 0.1$ ($N(0, 0.3 * \sigma(W_{org}))$ to the entire network.

4. Ensemble Models at the Parameter-Level: We create a single ensemble model by averaging the parameters of two consistent high-performing perturbed models. The PA score is generated from a single merged model. The models selected for fusion are the same ones used for ensembling at the score-level.

Table 3 provides the performance of these models (based on VGG19, ResNet101, and D-NetPAD architectures). The performance of perturbed and ensemble models is better than the original model on both datasets. The observation holds true for all three architectures. The perturbed models show an average improvement of 47.12% and 8.97%, the ensemble model at the score-level shows an improvement of 16.01% and 10.65%, and the ensemble model at the parameter-level shows an improvement of 43.58% and 9.25% on the LivDet-Iris-2017 and LivDet-Iris-2020 datasets, respectively. One major advantage of these perturbed models is that these models are created without any further training. Another advantage is that these high-performing perturbed models have reduced model size.

7. Summary and Future Work

We analyze the sensitivity of three DNN architectures (VGG19, ResNet101, and D-NetPAD) under three types of parameter perturbations (Gaussian noise manipulation, weight zeroing, and weight scaling). We apply the perturbations in two settings: modifying the weights across all layers and modifying weights layer-by-layer. We found that CNNs are generally less sensitive to a variant of weight zeroing, where low-magnitude weights are set to zero. From the layer-wise analysis, we observe that the CNNs are more robust to perturbations in later layers compared to the initial layers and Gaussian noise addition most negatively impacts the performance due to the zero-centered nature of weight distributions. Certain manipulations improve the performance over the original one. Based on these observations, we propose the use of an ensemble of models that consistently perform well on both LivDet-Iris-2017 and LivDet-Iris-2020 datasets. As future work, we will focus on finding the analytical optimum direction for weight perturbations. Additionally, the approach can be applied to other domains and tasks.

Acknowledgments

This research is based upon work supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via IARPA R&D Contract No. 2017 - 17020200004. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of ODNI, IARPA, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

References

[1] ISO/IEC 30107-1:2016: Information technology – Biometric Presentation Attack Detection – Part 1: Framework. https://www.iso.org/standard/53227.html. 3

[2] Nicholas Cheney, Martin Schrimpf, and Gabriel Kreiman. On the robustness of convolutional neural networks to internal architecture and weight perturbations. *arXiv*, abs/1703.08245, 2017. 1, 4

[3] Priyanka Das, Joseph Mcfiratht, Zhaoyuan Fang, Aidan Boyd, Ganghee Jang, Amir Mohammadi, Sandip Purnapatra, David Yambay, Sébastien Marcel, Mateusz Trokielewicz, et al. Iris liveness detection competition (LivDet-Iris)-the 2020 edition. *IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–9, 2020. 3

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 3

[5] Li Deng, Jinyu Li, Jui-Ting Huang, Kaisheng Yao, Dong Yu, Frank Seide, Michael Seltzer, Geoff Zweig, Xiaodong He, Jason Williams, et al. Recent advances in deep learning for speech research at microsoft. *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8604–8608, 2013. 1

[6] Li Deng and Yang Liu. *Deep learning in natural language processing*. Springer, Singapore, 2018. 1

[7] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 3

[8] Alhussein Fawzi, Seyed-Mohsen Moosavi-Dezfooli, and Pascal Frossard. Robustness of classifiers: From adversarial to random noise. *International Conference on Neural Information Processing Systems (NeurIPS)*, page 1632–1640, 2016. 1

[9] Siddhant Garg, Adarsh Kumar, Vibhor Goel, and Yingyu Liang. Can adversarial weight perturbations inject neural backdoors. *ACM International Conference on Information and Knowledge Management (CIKM)*, page 2029–2032, 2020. 1, 2

[10] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representations (ICLR)*, 2015. 1

[11] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. *International Conference on Neural Information Processing Systems (NeurIPS)*, page 1135–1143, 2015. 1

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. 1, 2, 3

[13] G. Huang, Z. Liu, L. v. d. Maaten, and K. Q. Weinberger. Densely connected convolutional networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2261–2269, 2017. 1, 2, 3

[14] Nikolaos Karianakis, Jingming Dong, and Stefano Soatto. An empirical evaluation of current convolutional architectures' ability to manage nuisance location and scale variability. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4442–4451, 2016. 1

[15] Carver Mead and Mohammed Ismail, editors. *Analog VLSI Implementation of Neural Systems*. The Kluwer International Series in Engineering and Computer Science. Kluwer / Springer US, 1989. 2

[16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 1

[17] Roman Novak, Yasaman Bahri, Daniel A. Abolafia, Jeffrey Pennington, and Jascha Sohl-Dickstein. Sensitivity and generalization in neural networks: an empirical study. *International Conference on Learning Representations (ICLR)*, 2018. 1, 2

[18] R. Polikar. Ensemble based systems in decision making. *IEEE Circuits and Systems Magazine*, 6(3):21–45, 2006. 6

[19] 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 (PAMI)*, pages 1–1, 2021. 2

[20] Renu Sharma and Arun Ross. D-NetPAD: An Explainable and Interpretable Iris Presentation Attack Detector. *IEEE International Joint Conference on Biometrics (IJCB)*, 2020. 3

[21] Hai Shu and Hongtu Zhu. Sensitivity analysis of deep neural networks. *ArXiv*, abs/1901.07152, 2019. 1

[22] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations (ICLR)*, 2015. 1, 2, 3

[23] Samuel L. Smith, Andrew Brock, Leonard Berrada, and Soham De. Convnets match vision transformers at scale, 2023. 3

[24] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *International Conference on Learning Representations (ICLR)*, 2014. 1

[25] Mateusz Trokielewicz, Adam Czajka, and Piotr Maciejewicz. Post-mortem iris recognition with deep-learning-based image segmentation. *Image and Vision Computing (IVC)*, 94:103866, 2020. 3

[26] Yu-Lin Tsai, Chia-Yi Hsu, Chia-Mu Yu, and Pin-Yu Chen. Formalizing generalization and adversarial robustness of neural networks to weight perturbations. *Advances in Neural Information Processing Systems (NeurIPS)*, 34:19692–19704, 2021. 1, 2

[27] University of Notre Dame. *The Notre Dame Contact Lens Dataset – NDCLD 2015*. available at: https://cvrl.nd.edu/projects/data. 3

[28] Tsui-Wei Weng, Pu Zhao, Sijia Liu, Pin-Yu Chen, Xue Lin, and Luca Daniel. Towards certificated model robustness

against weight perturbations. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):6356–6363, Apr. 2020. 1

[29] Lin Xiang, Xiaoqin Zeng, Yuhu Niu, and Yanjun Liu. Study of sensitivity to weight perturbation for convolution neural network. *IEEE Access*, 7:93898–93908, 2019. 1

[30] David Yambay, Benedict Becker, Naman Kohli, Daksha Yadav, Adam Czajka, Kevin W Bowyer, Stephanie Schuckers, Richa Singh, Mayank Vatsa, Afzel Noore, et al. LivDet iris 2017—iris liveness detection competition 2017. *IEEE International Joint Conference on Biometrics (IJCB)*, pages 733–741, 2017. 3

[31] Daniel S. Yeung, Ian Cloete, Daming Shi, and Wing W.Y. Ng. *Sensitivity Analysis for Neural Networks*. Springer Publishing Company, Incorporated, 2009. 1