Accelerating Spectral Normalization for Enhancing Robustness of Deep Neural Networks

Zhixin Pan and Prabhat Mishra
Department of Computer & Information Science & Engineering
University of Florida, Gainesville, Florida, USA

***Abstract*—Deep neural networks (DNNs) play an important role in machine learning due to its outstanding performance compared to other alternatives. However, DNNs are usually not suitable for safety-critical applications since DNNs can be easily fooled by well-crafted adversarial examples. To address this issue, spectral normalization (SN) technique was proposed to counter adversarial attacks, which ensures that the trained model has low sensitivity towards the disturbance of input samples. Unfortunately, this strategy requires exact computation of spectral norm, which is computation intensive and impractical for large-scale networks. In this paper, we introduce an acceleration technique for spectral normalization based on Fourier transform and layer separation. The proposed method provides DNNs with promising security protection while maintaining minimized time cost, which turns SN from a theoretically feasible approach to a practically useful framework. Experimental evaluation using autonomous systems demonstrates that our acceleration method is able to significantly improve both time efficiency (up to 60%) and model robustness (61% on average) compared with the state-of-the-art spectral normalization in real-world applications.**

***Index Terms*—Deep learning, adversarial attack, acceleration**

I. Introduction

Deep neural networks (DNNs) are widely used in machine learning field. Due to its outstanding performance in both supervised and unsupervised learning, DNNs can express or simulate a wide variety of intrinsic functionalities including classification, regression, Trojan detection [1], etc. The flexibility of DNNs also enables their different variations to be successfully employed in diverse applications [2]–[7]. However, the adversarial attack observed by Szegedy et al. [8] revealed the vulnerability of most existing neural networks against adversarial examples. The difference between adversarial samples and the original sample can hardly be distinguished by naked eyes, but will lead the model to make an incorrect prediction with high confidence. As shown in Figure 1, a human-invisible noise was added to input traffic sign image. While a pre-trained network can successfully recognize the original input as a stop sign, the same network will incorrectly classify it as a yield sign if the input is perturbed with well-crafted noise. In order to design robust DNNs, it is critical defend against adversarial attacks.

While there are many promising defense strategies, most of them are designed for specific attack algorithms, which severely restricts their applicability. Spectral normalization [9] has gained significant attention because it is algorithm-agnostic and can reduce DNNs' sensitivity to input perturbation. A major challenge with this method is that it provides a trade-off between computation time and numerical accuracy for computing the spectral norm of DNNs' weight matrix. Specifically, it applies one iteration of power method for each individual layer, which achieves poor accuracy in most cases. Moreover, it introduces high computation overhead when dealing with large convolution kernels. In reality, power iteration may not numerically converge to the desired result in specific scenarios, which limits its applicability.

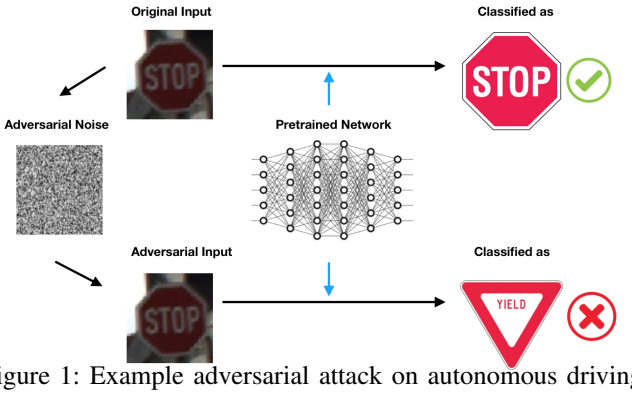


Figure 1: Example adversarial attack on autonomous driving: a stop sign miss-classified as a yield sign due to noise.

In this paper, we address this problem and propose a fast and dependable acceleration approach for spectral normalization. We have evaluated the effectiveness of our proposed approach using several DNN benchmarks. Specifically, this paper makes the following three important contributions.

1) Our work is the first attempt in utilizing spatial separation of convolution layers for regularized training.
2) We develop a fast approximate spectral norm computing scheme for convolution layers, which is based on Fourier transform and can be easily extended to other layers.
3) Experimental results using real-world dataset demonstrate our method improves networks' robustness against adversarial attack through extremely low attack success rate for bounded attack and high distortion for unbounded attack.

The remainder of this paper is organized as follows. We provide background on adversarial attack and spectral norm regularization in Section II. Section III describes our proposed method in detail. Section IV presents the experimental results. Finally, Section V concludes the paper.

This work was partially supported by the NSF grant CCF-1908131.

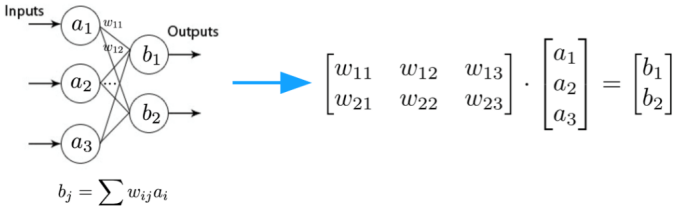
II. Background and Related Work

While DNNs are widely used in many applications, they are vulnerable to adversarial attacks. In order make DNNs robust against adversarial attacks, we need to understand the stability of neural networks. First, we briefly introduce the transformation of three types of layers in a DNN into matrix format. Next, we define a given model's stability using linear algebra concepts. Finally, we discuss related efforts to motivate the need for our proposed approach.

A. Three Types of Layers in DNNs

DNN consists of the following three types of layers: fully connected layer, convolution layer and activation functions.

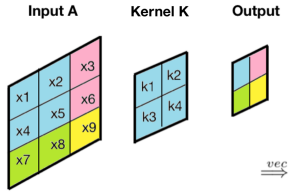
Linear Layer: In this layer, each output node is nothing but a weighted sum of inputs as shown in Figure 2. If we consider all equations and view it as a linear system, it can be represented in the form of matrix multiplication.



$$\begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

Figure 2: Fully connected layer in DNNs

Convolution Layer: Convolution layers are widely applied in computer vision tasks, where a convolution kernel K will slide along the surface of input feature map, and each weighted sum is stored into an entry of the output feature map. We can represent it as matrix multiplication between the input vector and a doubly block circulant matrix (which is called the convolution matrix of K). Figure 3 shows a trivial example to illustrate this idea.



$$\begin{bmatrix} x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 \\ x_7 & x_8 & x_9 \end{bmatrix} * \begin{bmatrix} k_1 & k_2 \\ k_3 & k_4 \end{bmatrix}$$

$$\overset{vec}{\Longrightarrow} \begin{bmatrix} k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 & 0 \\ 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 & 0 \\ 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 & 0 \\ 0 & 0 & 0 & k_1 & k_2 & 0 & k_3 & k_4 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_9 \end{bmatrix}$$

Figure 3: Convolution layer in DNNs

Activation Functions: Activation functions are inserted between consecutive network layers to induce nonlinearity to allow DNNs complete nontrivial tasks. They are usually selected as piecewise linear functions, such as ReLU, maxout, and maxpooling.

B. Stability of DNNs

Since we have already shown the close relationship between DNNs' layer and matrix multiplication, we can interpret adversarial attacks into the language of linear algebra. Assume that our model's interpreted matrix (we will call this *weight matrix* in the rest of this paper) is $W \in \mathbb{R}^{m \times n}$, and there is a well-crafted noise $\xi \in \mathbb{R}^n$ added to our input vector $x \in \mathbb{R}^n$, then we measure change of output by the following relative error $\epsilon \in \mathbb{R}$:

$$\epsilon = \frac{||W \cdot (x + \xi) - W \cdot x||_2}{||\xi||_2} = \frac{||W \cdot \xi||_2}{||\xi||_2}$$

Notice the spectral norm $\sigma(W)$ of matrix W is defined as:

$$\sigma(W) \triangleq \max_{\xi \neq \mathbf{0}} \frac{||W \cdot \xi||_2}{||\xi||_2}$$

Therefore, spectral norm of matrix W gives a tight upperbound of given layer's stability, and a small value of $\sigma(W)$ indicates this model's insensitivity to the perturbation of input x.

C. Related Work

Based on matrix representation of DNN' forward pass, recent research efforts have developed several countermeasures, but many of them failed to protect against strong attacks [10]–[14]. *Spectral Normalization* (SN) [9] is one of the most promising defense strategies because it is algorithm-agnostic approach and is able to enhance DNNs' robustness against adversarial attacks. During model training, the key idea of spectral normalization is to append the spectral norm of each layer to loss function as penalty term, so that it can reduce DNNs' sensitivity to input perturbation by minimizing every layer's spectral norm. The loss function is:

$$J = \frac{1}{N} \sum_{i=1}^{N} L(f(x_i), y_i) + \frac{\lambda}{2} \sum_{k=1}^{K} \sigma(W^k)^2$$

Here, x_i is a training sample, the label of x_i is denoted as y_i, the total number of training samples in a batch is denoted as N, L is the dissimilarity measurement and is frequently selected to be cross entropy or squared l_2 distance, $\sigma(W^k)$ is the spectral norm of k-th layer, K is the total number of layers, and $\lambda \in \mathbb{R}^+$ is a regularization factor.

The state-of-the-art method in [9] applied power iteration to approximate $\sigma(W^k)$, then perform *stochastic gradient descent* (SGD) to train the DNN, as shown in Algorithm 1.

Algorithm 1 State-of-the-art: Spectral Normalization (SN)

```
for each iteration of SGD do
    for k = 1 to K do
        for a sufficient number of iterations do
            Apply power iteration to compute σ(W^k)
            Add σ(W^k) to loss function
        end for
    end for
    Compute gradient of modified loss function
    Update parameters by back propagation
end for
```

This spectral normalization (SN) algorithm suffers from high computation complexity. Consider a convolution layer with a input channels, b output channels, and a convolution kernel in a size of $w \times h$. The corresponding weight matrix will be at least in a size of $b \times awh$. The exact computation of its spectral norm requires SVD decomposition, whose complexity

is $\boldsymbol{O}(min(m^2n, n^2m))$ for a $m \times n$ matrix, Therefore the time complexity in our task is $\boldsymbol{O}(min(a^2bw^2h^2, ab^2wh))$, which is infeasible to apply during network training. In order to make it run in a reasonable time, the authors in [9] use only a few iterations of power iteration method to approximate the spectral norms which leads to very coarse approximation. As a result, it significantly compromises its robustness goal. Our proposed method provides a fast approximation algorithm for spectral normalization through synergistic integration of layer separation and Fourier transformation as described in the next section.

III. Fast Approximate Spectral Normalization

Figure 4 shows an overview of our proposed framework for *fast spectral normalization* (FSN). The major steps of our framework is outlined in Algorithm 3. Our proposed framework consists of two strategies to accelerate training process: *layer separation* and *Fourier transform*. The remainder of this section describes these strategies in detail.

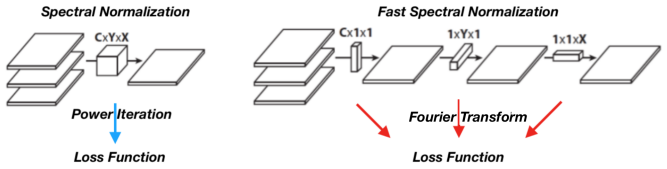


Figure 4: Comparison of our proposed framework with the state-of-the-art. We utilize layer separation to decompose large and multi-dimension layers into 1-D layers, then apply Fourier transform on each of the kernels.

A. Layer Separation

The reason for computing spectral norm being so expensive is the size of weight matrix. Here we apply *layer separation* to reduce the time complexity for this task. First, a 2-D filter kernel K is said to be $separable$ if it can be expressed as the outer product of one row vector $\boldsymbol{r}$ and one column vector $\boldsymbol{c}$. But vectors are special cases of matrix, and their outer product is equivalent to their convolution. Then due to the associativity of convolution:

$$K = \boldsymbol{r} \times \boldsymbol{c} = \boldsymbol{r} * \boldsymbol{c}$$
$$\Rightarrow A * K = A * (\boldsymbol{r} * \boldsymbol{c}) = (A * \boldsymbol{r}) * \boldsymbol{c}$$

A famous example will be the Sobel kernel, which is widely used to approximate the gradient value of image brightness function:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}$$

Once the given convolution layer is separable ($\text{rank}(W) = 1$), we can always decompose it into 2 1-D layers but achieve the same effect. This time we have 2 consecutive convolution layers, and their kernel size will be $w \times 1$ and $1 \times h$. We can compute each layer's spectral norm separately and then append them to the loss function together.

What about inseparable layers? In this case, our method utilize the SVD decomposition of kernel filter to transfer it into its low-rank approximation. Here, we are performing SVD decomposition on kernel matrix instead of the doubly block circulant matrix. Kernel matrix encountered in modern CNNs are usually in the form of 3×3 or 5×5, time cost for SVD decomposition on these tiny matrices is trivial. Also, applying low-rank approximation of kernel filter is reasonable when it comes to real situations. In CNN applications, especially in computer vision areas, many frequently used feature extraction kernels like edge detection filters are trained to have limited number of dominant eigen directions, where the largest singular value is far larger than the inferior ones, one extreme example is the Sobel kernel mentioned above. Under this circumstance, the deviation induced by low-rank approximation can be neglected.

B. Fourier Transform

It is a well known fact that the spectral norm of a 2D convolution kernel K is exactly the largest eigenvalue of $A^\top A$, where A is the corresponding convolution matrix of K. In the baseline method illustrated in Algorithm 1, power iteration method was applied. But power method, as a fundamental numerical method for computing eigenvalues, usually takes too many iterations until acceptable accuracy is obtained. Also, it assumes the weight matrix has an eigenvalue that is strictly greater in magnitude than the others, and the initial random vector should contain an nonzero component in the direction of a dominant eigenvector. If the above assumptions fails, the power method may not converge! Instead, we apply the following theorem [15] to calculate the spectral norm of a convolution kernel fast.

Theorem 1: For any convolution matrix formed by kernel K, the eigenvalues of the convolution matrix are the entries of 2D Fourier transform of K, and its singular values are their magnitudes.

Proof: Assume K be a kernel matrix, and A is the convolution matrix of K (as described in Figure 3). Now the task is to determine the singular values of a doubly block circulant matrix A. First, we define

$$Q \triangleq \frac{1}{n}(F \otimes F)$$

where F is the Fourier matrix. To complete the proof, we use the following lemma.

Lemma 1: For any doubly block circulant matrix A, the eigenvectors of A are the columns of Q, and Q is unitary. (J.Toriwaki(1989) in [16])

Based on the above lemma, we know A can always be decomposed into its eigenvalue decomposition $A = Q * DQ$ where D is a diagonal matrix. Now we show that A is normal:

$$AA^\top = AA^* = Q^*DQQD^*Q$$
$$= Q^*DD^*Q = Q^*D^*DQ = Q^*D^*QQ^*DQ$$
$$= A^*A = A^\top A$$

Since the singular values of any normal matrix are the magnitudes of its eigenvalues (Johnson (2012), page 158 [17])

and we have shown that A is normal, by applying the Lemma 1 we complete the proof. ■

Therefore, we can calculate the spectral norm of target convolution kernel by Fourier transform, and this approach is depicted in Algorithm 2.

Algorithm 2 Algorithm for computing spectral norms

Input: convolution kernel K
Output: σ, the spectral norm respected to K.
1: $k \leftarrow$ 2D Fourier transform of K
2: $k' \leftarrow$ set all the entries of k to zero except the one with maximum absolute value
3: $K' \leftarrow$ 2D inverse Fourier transform of k'
4: $\sigma \leftarrow$ largest entry inside K'.

C. Activation Functions

By exploiting layer separation and Fourier transform, the computation of spectral norm for linear and convolution layers can be solved efficiently as shown in Algorithm 2. When it comes to activation layers, there is no inherent way to represent activation layers in matrix format. Activation functions are deployed in neural networks to induce nonlinearity, therefore it is impossible to view an activation function as a linear transformation.

To perform our method with activation functions, instead of parsing it into matrix and compute spectral norm, we take their *Lipschitz constants* into consideration. A function f defined on $\mathbf{X}$ is said to be $K-$Lipschitz if

$$\forall x_1, x_2 \in \mathbf{X}, |f(x_1) - f(x_2)| \leq K|x_1 - x_2|$$

The smallest possible constant K for f is also called *Lipschitz norm*, which reflects how expansive function f is. The Lipschitz norm upper bounds the relationship between input perturbation and output variation for a given distance. Notice the similarity between the definition of spectral norm and Lipschitz norm. Actually, spectral norm of matrix W is essentially the Lipschitz norm of function f if $f(x) = Wx$.

It has been proven that most activation functions such as ReLU, Leaky ReLU, SoftPlus, Tanh, Sigmoid, ArcTan or Softsign, as well as max-pooling, are *short maps* [18], i.e, they have a Lipschitz constant equal to 1. We refer the reader to [19] for detailed proof on this subject. As a result, the output variation induced by activation layers is already restricted by a constant number. Therefore, there is no need to spare extra effort to append regularization terms for activation functions to loss function, since it will not get changed during training. Moreover, the constant number is 1, which means activation functions will neither expand or contract the variation of layer outputs. In other words, we can omit activation layers when evaluating the stability of DNNs.

D. Complexity Analysis

The complete framework for approximate spectral normalization is depicted in Algorithm 3. For our target problem, based on the discussion in Section II-C, we decompose the $b \times awh$ weight matrix into 2 matrices, $b \times aw$ and $b \times ah$ for each. Then we apply Fourier transform on each separated matrix. In our experiment *fast Fourier transform* (FFT) is applied with $\boldsymbol{O}(mnlog(mn))$ time complexity for $m \times n$ matrix. So the proposed method's complexity is $O(abw\log(abw) + abh\log(abh)))$.

Algorithm 3 Proposed: Fast Spectral Normalization (FSN)

for each iteration of SGD **do**
  Compute the gradient of general loss function as usual
  for $k = 1$ to K **do**
    if Convolution Layer **then**
      Perform layer separation
      Form the corresponding convolution matrix
    end if
    if Linear Layer **then**
      Form the corresponding weight matrix
    end if
    Perform **Algorithm2** to compute $\sigma(W^k)$
    Add $\sigma(W^k)$ to loss function
  end for
  Update parameters using modified gradient
end for

IV. Experiments

We evaluated the effectiveness of our optimized spectral norm regularization framework to confirm its time-efficiency and ability of enhancing DNNs' robustness.

A. Experiment setup

Experiments were conducted on a host machine with Intel i7 3.70GHz CPU, 32 GB RAM and RTX 2080 256-bit GPU. We developed code using Python for model training. We used PyTorch as the learning library. For adversarial attack algorithms, we utilized the *Adversarial Robustness 360 Toolbox* (ART) [20]. For experimental evaluation, we considered the following two DNN configurations applied in real-world autonomous driving systems as experiment subjects:

1) A VGG16 classifier for *German Traffic Sign Benchmark* (GTSB) dataset [21].
2) A Lenet-5 network for *Udacity Self-Driving* (USD) dataset [22].

For each setting we compare the following three approaches:

1) **Normal**: Ordinary training without regularization, which is considered as control group.
2) **SN**: State-of-the-art approach (Algorithm 1) with spectral normalization [9].
3) **FSN**: Our proposed approach (Algorithm 2) with fast and robust spectral normalization.

As for result evaluation, we first evaluated the functionality of them by reporting their accuracy and training time. Next, both bounded and unbounded adversarial attacks were deployed to test their robustness. Finally, we tuned the regularizer factor λ for FSN to demonstrate its stability across a wide variety of different λ values.

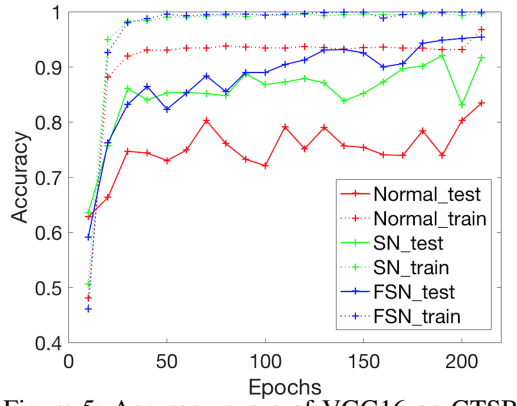
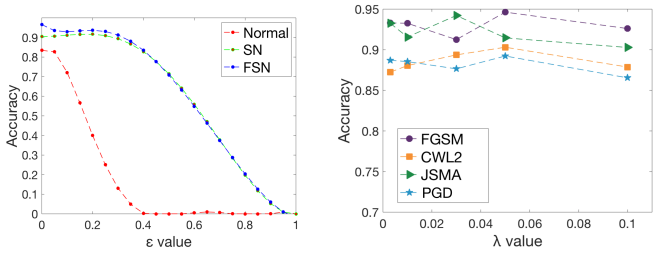


Figure 5: Accuracy curve of VGG16 on GTSB



(a) Performance of three models towards unbounded FGSM attack

(b) FSN's performance with different λ and attack algorithm

Figure 6: Performance with varying hyperparameter (GTSB)

B. Case Study: German Traffic Sign Benchmark (GTSB)

We train VGG16 on GTSB dataset. GTSB is a large and reliable benchmark which contains more than 50000 traffic sign image samples over 43 classes. VGG16 [23] contains 16 hidden layers (13 convolution layers and 3 fully connected layers). One advantage of VGG16 is the replacement of larger convolution kernels ($11\times11, 7\times7, 5\times5$) with consecutive 3×3 convolution kernels, which makes it coincidentally suitable for our framework since small size of convolution kernel implies less time cost for low-rank approximation. We choose a mini-batch size of 64. For each setting, we train the network with 200 epochs. For every 10 epochs, we randomly shuffle and split 80% as training set and 20% as test set, and then report the training time along with test performance.

Table I: Training time and test accuracy of VGG16

Methods	Time(s/epoch)	Best Test Accuracy(%)
Normal	14.5053	83.45
SN	29.1126	91.72
FSN	18.1345	95.46

Table I indicates the basic functionality performance of three training methods. We also plot the training curve for accuracy updates in Figure 5. In general, all three methods possess good training accuracy. The Normal approach achieved the fastest speed but when it comes to test accuracy, it lagged behind the other two due to spectral normalization's ability to enhance model's generalizability [9]. SN improves the test accuracy by 8.27% but leads to highest time cost. FSN further improves the test accuracy by 3.66% while drastically reduces the average training time by 60.5% compared to SN.

Table II: Bounded Attack on GTSB

Methods	N/A	FGSM (ε = .1)	CW-L_2 (c = .01)	JSMA ($\theta, \gamma = .1, 1$)	DeepFool $\varepsilon = 1e-6$
Normal	83.45	61.04	45.62	77.58	40.06
SN	91.72	88.25	76.31	88.23	69.30
FSN	95.46	92.92	87.05	91.57	88.94

Table II shows the performance of three models against bounded attacks. We choose four different attack algorithms: FGSM [13], CW-L_2 [24], JSMA [25] and DeepFool [11]. For consistency $\lambda = 0.01$ for both SN and FSN here. The hyperparameters for attacks are provided in the table.

As we can see, our proposed method (FSN) provides the best robustness. The Normal and SN method appeared fragile in the face of powerful attacks like CW-L_2 and DeepFool, while FSN still retained an acceptable accuracy. For lightweight attacks, especially gradient-based ones (like FGSM), FSN is almost unaffected. For unbounded attack, we applied unbounded incremental ε value in FGSM from 0 to 1. It starts to break detection (accuracy $< 50\%$) as presented in Figure 6(a), but the bisection parameter value of FSN($\varepsilon = .64$) has to be nearly three times large of that for Normal method($\varepsilon = .21$). Finally, FSN's broad robust performance across a wide range of hyperparameter λ is demonstrated in Figure 6(b) where we plot accuracy under all attacks with λ varying from 0.01 to 0.1. The average accuracy here is 91.46% with standard deviation of 0.0729. In the worst case, model trained with FSN still performed well with accuracy above 85%.

C. Case Study: Udacity Self-Driving (USD)

We trained Lenet-5 on USD's steering angle image dataset which has more than 5000 autonomous vehicle's real-time screenshot along with labels indicating the corresponding "Left", "Right" or "Center" prediction for next wheel steering operation. Compared with VGG16, Lenet-5 is a lightweight network with only five convolution layers and one fully connected layer. The training process consists of 500 epochs with a mini-batch size of 128. For every 50 epochs, we randomly shuffle sample and split 80% as training set and 20% as test.

Table III: Training time and test accuracy of Lenet-5

Methods	Time(s/epoch)	Best Test Accuracy(%)
Normal	0.13369	99.71
SN	0.25870	98.57
FSN	0.18284	99.43

Table III and Figure 7 present performance results. Again, Normal method provides the fastest training speed, while FSN is nearly 41.4% faster than SN. Note that the improvement of speed here is not as high as we got from GTSB case, and we consider this difference of acceleration amplitude is caused by more convolution layers involved in VGG16 than Lenet-5. For accuracy, due to Lenet-5's excellent capability in image recognition, all three models achieved high level of accuracy.

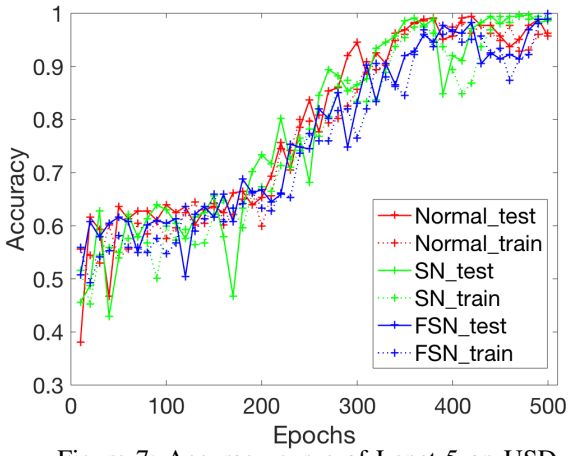
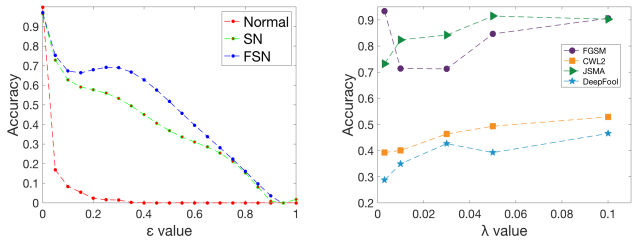


Figure 7: Accuracy curve of Lenet-5 on USD

The robustness is evaluated against four attack algorithms as shown in IV. As expected, Normal method behaved extremely vulnerable. Even in $\varepsilon = 0.1$ case, a 8.3% accuracy against FGSM attack was obtained. FSN refined it to 71.4% which gives an 61.1% average improvement on accuracy. When it comes to unbounded attack as depicted in Figure 8(a), the Normal method's outcome immediately drops below 50% when $\varepsilon = 0.1$, while SN and FSN extended it to 0.4 and 0.5, respectively. Figure 8(b) shows the accuracy after applying various regularization factor of λ. The average accuracy is 62.63% with standard deviation of 0.2252. If we consider only CW-L_2 and DeepFool, the average drops to 41.99% with standard deviation of 0.0713.

Table IV: Bounded Attack on USD

Methods	N/A	FGSM (ε = .1)	CW-L_2 (c = .01)	JSMA ($\theta, \gamma = .1, 1$)	DeepFool $\varepsilon = 1e-6$
Normal	99.71	8.3	5.62	17.58	4.06
SN	98.57	69.9	34.34	78.23	29.30
FSN	99.43	71.4	40.05	82.34	34.94



(a) Performance of three models under unbounded FGSM attack

(b) FSN's performance with different λ and attack algorithm

Figure 8: Performance with varying hyperparameters (USD)

V. Conclusion

Adversarial attacks are vital threats preventing DNNs' adoption in safety-critical applications. In this paper, we investigated DNN layers' properties and proposed FSN as an acceleration algorithm for achieving spectral normalization. Such a regularization DNN training technique made three important contributions. (1) FSN utilized the spatial separation of convolution layers as well as Fourier transform to drastically (up to 60%) reduce training time compared with traditional spectral normalization. (2) FSN is algorithm-agnostic, easy to implement, and applicable across a wide variety of networks. (3) Experimental evaluation demonstrated that models trained with FSN provide significant improvement in robustness for bounded, unbounded as well as transferred threats, which can protect applications from various adversarial attacks. Compared with the state-of-the-art, our approach ensures that the model works correctly unless the adversarial samples are drastically different from the original samples. Our proposed acceleration turns SN from a theoretical idea into an practical training approach, which is expected to play a crucial role in improving the robustness of DNNs against adversarial attacks.

References

[1] Z. Pan and P. Mishra, "Automated test generation for hardware trojan detection using reinforcement learning," in *ASPDAC*, 2021, pp. 408–413.

[2] Z. Pan, J. Sheldon, and P. Mishra, "Hardware-assisted malware detection using explainable machine learning," in *ICCD*, 2020, pp. 663–666.

[3] Z. Pan, J. Sheldon, C. Sudusinghe, S. Charles, and P. Mishra, "Hardware-assisted malware detection using machine learning," in *DATE*, 2021.

[4] S. Charles, A. Ahmed, U. Y. Ogras, and P. Mishra, "Efficient cache reconfiguration using machine learning in noc-based many-core cmps," *ACM TODAES*, vol. 24, no. 6, pp. 1–23, 2019.

[5] A. Ahmed, Y. Huang, and P. Mishra, "Cache reconfiguration using machine learning for vulnerability-aware energy optimization," *ACM Trans. on Embedded Computing Systems*, vol. 18, no. 2, pp. 1–24, 2019.

[6] K. Rahmani and P. Mishra, "Feature-based signal selection for post-silicon debug using machine learning," *IEEE TETC*, 2017.

[7] K. Rahmani, S. Ray, and P. Mishra, "Postsilicon trace signal selection using machine learning techniques," *TVLSI, 25(2)*, pp. 570–580, 2016.

[8] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," 12 2013.

[9] Y. Yoshida and T. Miyato, "Spectral norm regularization for improving the generalizability of deep learning," 05 2017.

[10] N. Carlini and D. A. Wagner, "Towards evaluating the robustness of neural networks," *CoRR*, vol. abs/1608.04644, 2016.

[11] S.-M. Moosavi-Dezfooli, A. Fawzi, and P. Frossard, "Deepfool: a simple and accurate method to fool deep neural networks," *CVPR*, 11 2016.

[12] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *CoRR*, vol. abs/1605.07277, 2016.

[13] I. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 01 2015, pp. 1–10.

[14] A. Kurakin *et al.*, "Adversarial examples in the physical world," in *5th International Conference on Learning Representations*, 2017.

[15] H. Sedghi, V. Gupta, and P. M. Long, "The singular values of convolutional layers," 2018.

[16] J. Toriwaki and H. Yoshida, *Fundamentals of Three-dimensional Digital Image Processing*. Springer, 2009.

[17] R. A. Horn and C. R. Johnson, *Matrix Analysis, 2nd Ed.* Cambridge University Press, 2012.

[18] A. Virmaux and K. Scaman, "Lipschitz regularity of deep neural networks: analysis and efficient estimation," in *NIPS*, 2018.

[19] Y. LeCun, *et al.*, "Deep learning," *Nature*, vol. 521, no. 7553, 2015.

[20] M.-I. Nicolae *et al.*, "Adversarial robustness toolbox v1.0.1," *CoRR*, vol. 1807.01069, 2018. [Online]. Available: https://arxiv.org/pdf/1807.01069

[21] J. Stallkamp *et al.*, "The German Traffic Sign Recognition Benchmark: A multi-class classification competition," in *IJCNN*, 2011.

[22] "The udacity open source self-driving car project." [Online]. Available: https://github.com/udacity/self-driving-car

[23] A. Pedraza, J. Gallego, S. Lopez, L. Gonzalez, A. Laurinavicius, and G. Bueno, "Glomerulus classification with convolutional neural networks," in *Medical Image Understanding and Analysis*, 2017.

[24] N. Carlini and D. Wagner, "Adversarial examples are not easily detected: Bypassing ten detection methods," in *WAIS*, 2017, pp. 3–14.

[25] R. Wiyatno and A. Xu, "Maximal jacobian-based saliency map attack," *CoRR*, vol. abs/1808.07945, 2018.