# DEFENDING AGAINST ADVERSARIAL ATTACKS ONE LAYER AT A TIME

A PREPRINT

**Yuval Epstain Ofek**
Electrical Engineering
The Cooper Union
New York, NY 10003
epstain@cooper.edu

**Jason Kurian**
Electrical Engineering
The Cooper Union
New York, NY 10003
kurian@cooper.edu

December 17, 2020

**ABSTRACT**

Research on adversarial attacks have brought along a new security threat to using deep neural networks. Common defenses against such attacks vary between training using adversarial attacks and pre-classifier defense mechanisms. Our work proposes a new method to defend against adversarial attacks, inserting defensive layers inside the network. We formulate two such defenses, which we call defensive layers, which we show can negate the effect of adversarial attacks. We further show that the ideal place in the network to insert these defenses is not at the beginning of the network, as prior work suggests, but rather deeper inside the network. Our code is publicly available at: https://github.com/yuvalofek/DefensiveLayer.git

***K*eywords** Adversarial Attacks · Grey-box setting · Adversarial Defense · Defensive Layer · Intra-model Defense

## 1 Introduction

With the advent of deep neural networks (DNNs) and their increasingly widespread use, the security of these networks becomes a pressing concern. While a DNN may achieve high accuracy on natural data, Szegedy et al. [10] suggest that adversarial attacks - malicious modifications or perturbations to the input data - may intentionally force the DNN to misclassify its inputs. For instance, adversarial attacks have been shown to fool DNNs systems for face detection, and autonomous vehicles [11][1]. Such attacks are an ongoing threat to security-sensitive DNNs and may trigger disastrous consequences, such as fooling cars into misclassifying traffic signs and causing irreparable accidents [5].

Our attempt to defend against such attacks is based on an intermediary layer approach, in which the defense mechanisms used to counteract adversarial misclassifcation are inserted within the classifier model itself. Through this method we aim to remove adversarial noise affecting intra-model features that a pre-classifier defense would not account for.

Our main contributions to this field of study are:

1. We propose two types of defensive layers that implement wavelet denoising on inputs composed of more than three channels.
2. Through empirical evaluations we show the effects of inserting intra-model defense layers one at a time or simultaneously on the classifier's accuracy for clean and perturbed images. We identify where and where not in the model the layers should be inserted to best improve accuracy.

The paper will be structured as follows. We first discuss related work on adversarial attacks and defenses (**Section 2**), followed by a description of our specific intra-model defense approach (**Section 3**). We then present our empirical processes and results (**Section 4**) before discussing the implications of what was observed (**Section 5**).

## 2 Prior Works

### Adversarial Attacks

A common adversarial attack used to perturb images is the Fast Gradient Sign Method (FGSM) introduced by Goodfellow et al. [2]. FGSM adds noise, the magnitude of which is governed by some epsilon, to each pixel in an image to maximize the loss function, hence going in the opposite direction of the gradient. The images are generated as:

$$x_{adv} = x_c + \epsilon * sign(\nabla(L(x_c, y_c; \theta))) \tag{1}$$

Another common attack method is DeepFool proposed by Moosavi-Dezfooli et al. [4]. This attack aims to minimize the L2 norm of a given image and an another class until the targeted image crosses the decision boundary and is misclassified. The process of generating the images is based on an iterative linearization of the classifier to generate minimal perturbations that will result in misclassification.

### Defenses

Deep learning defense mechanisms established to combat adversarial attacks can generally be grouped into three categories. In adversarial training, the classifer is trained on perturbed images to improve robustness, though this does not defend against attacks not represented in the training dataset. Other methods modify the network architecture to reduce the magnitude of the gradients around the input points so that it is difficult to generate adversarial examples that successfully cause misclassification. Lastly, another defense method is to detect the adversarial examples before they are fed into the model and denoise them.

Mustafa et. al. [6] proposed an adversarial defense combining super-resolution and wavelet denoising to map *off-the-manifold* adversarial images onto the natural image manifold to restore their classification towards correct classes. They used enhanced deep super resolution (EDSR) [3] to enhance pixel resolution and remove high frequency adversarial patterns. After applying super-resolution to the images, they put them through wavelet denoising for an added suppression technique of adversarial noise patterns. We base our defensive approach off of wavelet denoising.

Wavelet denoising can be used as an adversarial defense by removing the noise in adversarial samples. The Discrete Wavelet Transform of real-world images have large coefficients representing the important features in the images, so by setting a threshold on the wavelet coefficients, wavelet denoising effectively removes adversarial noise from the images.

## 3 Methods

In contrast to prior work, which aimed at defending a DNN by pre-pending a defensive mechanism, we propose a method that inserts the defense inside the network itself. Intuitively, the adversarial attacks have been constructed to alter the output of the network using a loss that has been propagated back to the input layer. This loss takes into account the entire network, so defenses in between network layers hold potential in mitigating adversarial attack concern despite not having been thoroughly explored in literature. We propose to insert "defense layers" dynamically within the classifier network to further improve the accuracy on adversarial images.

### Algorithmic Description

We introduce two stochastic defense layers utilizing denoising in Algorithm 1 and Algorithm 2. In Algorithm 1, we repeatedly select three channels at random, denoise the channels, and return the denoised channels to their previous positions, relying on an external hyperparameter to determine the number of times to repeat the process at each layer pass. In Algorithm 2, we force the channel selection to include almost all of the channels in the denoising procedure by doing a sweep over the channels. We first reorder the channels and divide them into groups of threes, then we denoise each of the groups, and finally put the channels back in their original positions (before the reordering). This both removes the additional hyperparameter introduced earlier and also ensures a channel is not denoised multiple times. We used wavelet denoising in our implementations

This defense technique is intended for fully trained models, as the denoising layers are non-differentiable. Intuitively, this is both good and bad. It is good because no adversarial attack can be created for the model after the defense is in

place, as adversarial attacks require gradient information to propogate to the input. On the other hand, adding these layers inside a trained model might cause an accuracy drop in non-adversarial settings.

**Algorithm 1:** Repeated Random 3-Channel Denoising

**Require:** Previous layer output $x$
**Require:** $N_{den}$; hyperparameter
1. **for all** $N_{den}$ **do**
2. Select 3 channels from $x$ called $x_i$
3. Perform denoising on $x_i$ to obtain $x_i'$
4. Replace $x_i$ in $x$ with $x_i'$
5. **end for**

**Algorithm 2:** Random Channel Denoising Sweep

**Require:** Previous layer output $x$
1. Randomly order $channels$ in $x$ in groups of 3 called $groups$ *(Note: Omitting any remaining channels)*
2. **for all** $x_i$ in $groups$ **do**
3. Perform wavelet denoising $x_i$ to obtain $x_i'$
4. Replace $x_i$ in $x$ with $x_i'$
5. **end for**

## 4 Experiments

### Models and Datasets

The VGG19 model [9] pretrained on ImageNet [8] was used as the base classification network for our analysis. Since the model was pretrained, we only needed a test dataset for our analysis. Do to lack of computational power, we chose to select images from two classes in Imagenet: ships, and bikes for our test set. These images were then used as a starting point in generating attacks using the FGSM and DeepFool techniques. Implementations for these attacks can be found in Foolbox [7].

### Intermediate Layers

We first compare the two defensive layers we formulated, the results of which are seen in Figure 1.The random sweep algorithm was decided to be the superior method as it had a slightly higher average test accuracy; moreover, the possibility of repeatedly denoising the same channels in the other algorithm was deemed computationally wasteful and thus unnecessary. The rest of the experimentation was performed only with the layer implementing the random sweep algorithm.

### Single Layer Insertions

We then compare the relative accuracy of the model when inserting the layer at different depths inside the network, as can be seen in Figure 2a. The depth of layer insertion is defined as the existing position of the layer in the model that the denoising layer was appended to after. The accuracy of the classifier for each test configuration was averaged over five runs for robustness.

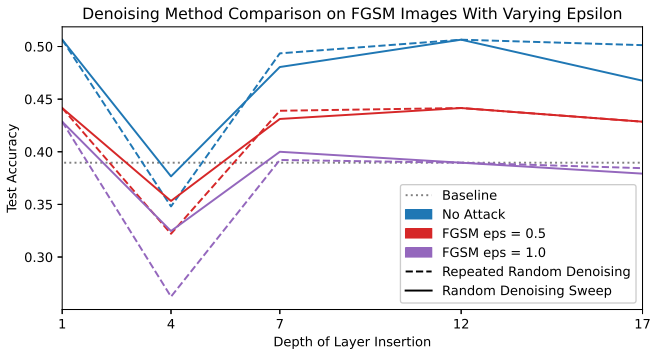
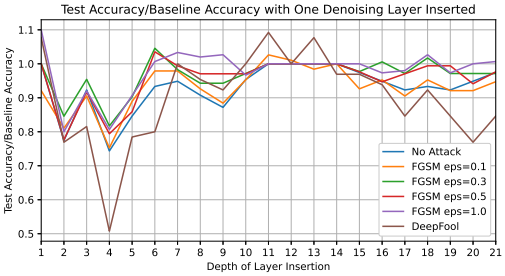
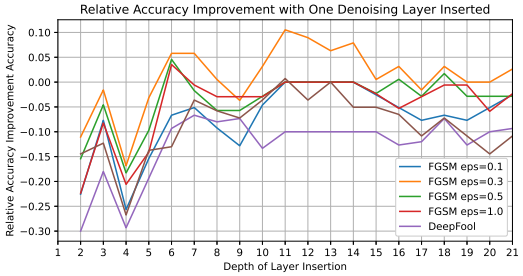



Figure 1: Comparison between repeated random channel denoising and random channel denoising sweep




(a) Relative accuracy ($accuracy/baseline$) per defense layer insertion depth for different adversarial attacks




(b) Relative accuracy improvement ($(accuracy - accuracy_{no-attack})/baseline$) per defense layer insertion depth for different adversarial attacks

Figure 2: Single Layer Insertions

There are some general patterns that emerge across the attack types. After a steep decline in accuracy from an early insertion to a depth of 4, the accuracies then peak somewhere between depths 11 and 14, until declining again in the remaining positions. The accuracies generally reach a trough when the denoising layers are added after a max pooling layer, which are layers 4, 9, 14, 17, and 20.

We isolate the effects of the layer on the model's typical accuracy (i.e. the accuracy when no attacks are present) by taking the difference from the no attack accuracy. The resulting trends can be seen in Figure 2b.

We now observe a slightly different trend than before. In general, we see that the accuracy stays pretty consistent with layer width. We still see the dip at around the 4th layer, but now we see that this is also linked to a general drop in accuracy at the earlier layers. We also note that the peak observed in the earlier at around the 11-14 layers still remains.

**Multiple Layer Insertions**

We tested how the classifier behaved with the addition of multiple denoising layers at once. Figure 3 shows the relative accuracies of the model as we increased the number of layers inserted. The depths at which the layers were inserted were determined by the test accuracies of the model given a single layer insertion at that depth, shown in Figure 2a, sorted in descending order. For example, in the trial where two layers were inserted at once, those two layer positions achieved the first and second best test accuracies in the single insertion experiment.

The classification accuracy on the original images remained essentially constant through up to five layers inserted at once. Adding three denoising layers at once only improves the test accuracy for images attacked with DeepFool, while the addition of two layers improves the test accuracy for FGSM-attacked images perturbed with an epsilon of 0.5.

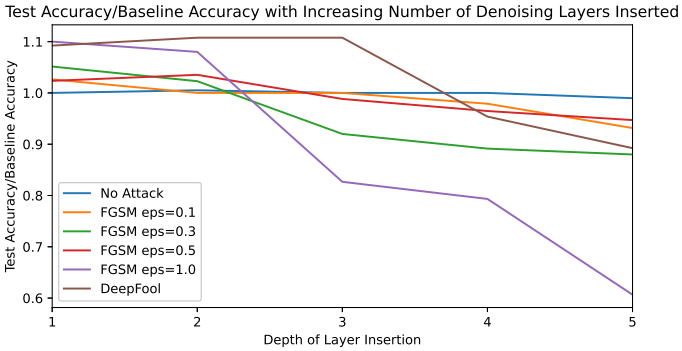



Figure 3: Relative accuracy ($accuracy/baseline$) per number of defense layer insertions for multiple adversarial attacks

## 5 Discussion

Our results may tell us to re-evaluate assumptions made by prior works in adversarial defense, which opted for a singular defensive mechanism. Our results suggest that a network has certain depths that lend themselves toward adversarial defense, which begins to demonstrate the idea of what we coin as 'layer susceptibility'— how much a given layer contributes to the resulting miss-labeled output. Furthermore, we have shown that adding a defense to the beginning of the network has been less useful than adding it at a layer point inside the network.

The results seen in Figure 2a are also thought inducing. A consistent pattern of relative accuracy dips are observed around the fourth layer, which is the first maxpooling layer. We believe that this is due to the maxpooling layer having a discretizing affect on the data, so that having the denoising layer after the maxpooling layer veers the results farther away from the training data distribution, confusing the model. Figure 2a also suggests the central convolutional block, consisting of four convolutional layers (10-13) and a max pooling layer (14), is where intermediary denoising is most beneficial. Denoising at depths further along is essentially too far into the model for the subsequent trained layers to recover the intended classification.

## 6 Conclusions

Adversarial attacks pose serious security risks to deep learning applications and may cause significant harm to individuals and society. Therefore, designing deep learning defense mechanisms that are robust to a variety of different attacks and attack magnitudes is critical. In this work, we proposed two post-hoc defensive mechanisms that are adaptable to any convolutional layer application and do not require any further setup or training. We showed that our defensive mechanism can negate some of the effects of adversarial attacks. We further showed how the position of the inserted defensive layer affects the network's final accuracy and suggested an interpretation that some point along the network are more susceptible to the adversarial attack than others. Lastly, we studied the effects of adding more than one defense layer to a given network and showed that the optimal number of defensive layers is one in most scenarios.

## 7 Acknowledgements

We would like to thank Professor Curro and the ECE-472 class for their help and suggestions while completing this project.

## References


[1] Y. Deng, X. Zheng, T. Zhang, C. Chen, G. Lou, and M. Kim. An analysis of adversarial attacks and defenses on autonomous driving models, 2020.

[2] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples, 2015.

[3] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee. Enhanced deep residual networks for single image super-resolution, 2017.

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

[5] N. Morgulis, A. Kreines, S. Mendelowitz, and Y. Weisglass. Fooling a real car with adversarial traffic signs, 2019.

[6] A. Mustafa, S. H. Khan, M. Hayat, J. Shen, and L. Shao. Image super-resolution as a defense against adversarial attacks. *IEEE Transactions on Image Processing*, 29:1711–1724, 2020. ISSN 1941-0042. doi: 10.1109/tip.2019.2940533. URL `http://dx.doi.org/10.1109/TIP.2019.2940533`.

[7] J. Rauber, W. Brendel, and M. Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models, 2018.

[8] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[9] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[10] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks, 2014.

[11] F. Vakhshiteh, R. Ramachandra, and A. Nickabadi. Threat of adversarial attacks on face recognition: A comprehensive survey, 2020.

# 8 Appendix I - Additional Figures

The results of the single and multiple layer insertion experiments are graphed separately by attack type and epsilon in this section. The data from these graphs were aggregated to produce the graphs shown in Figures 2a, 2b, and 3.

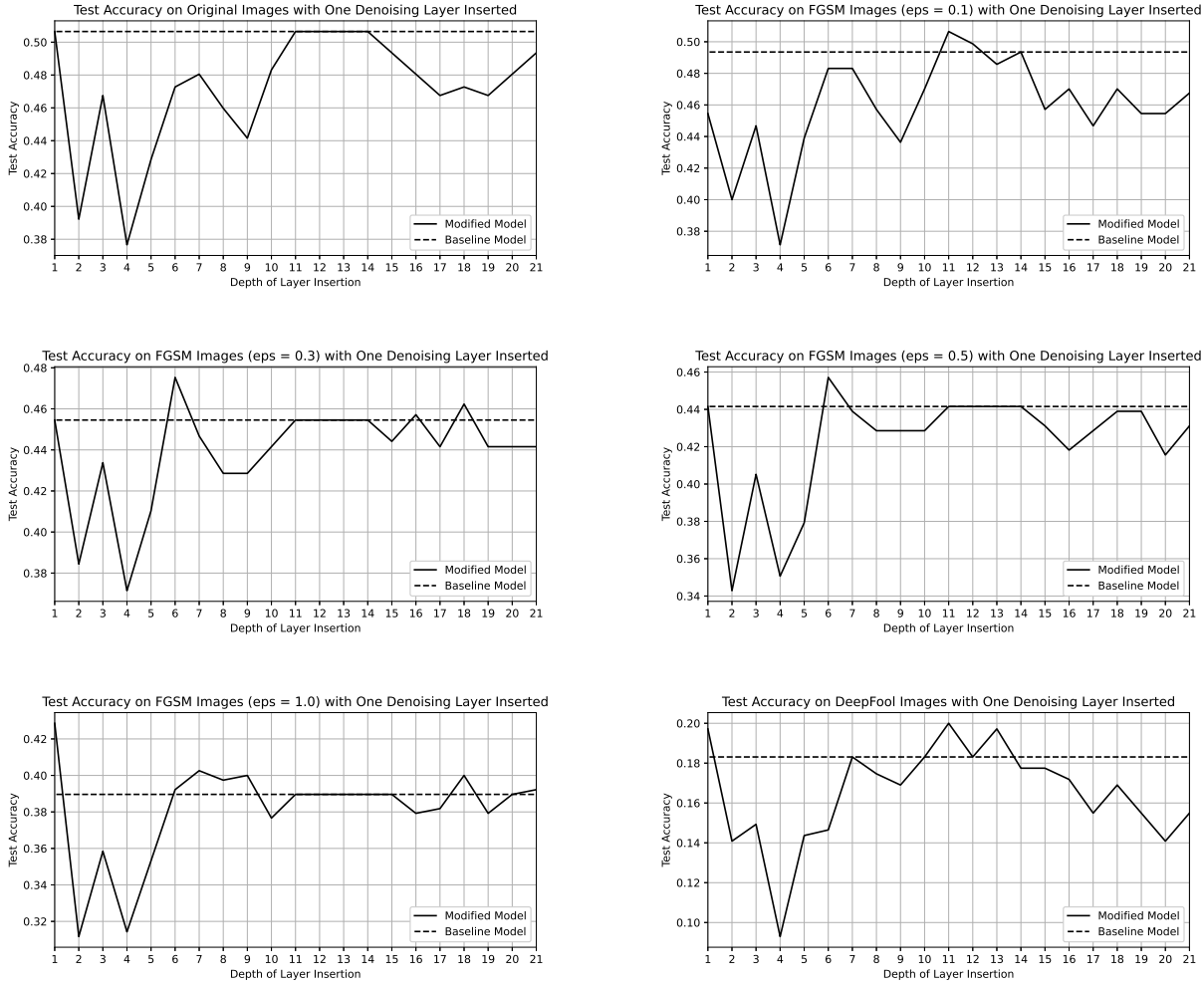



Figure 4: Single Layer Insertion Experiments

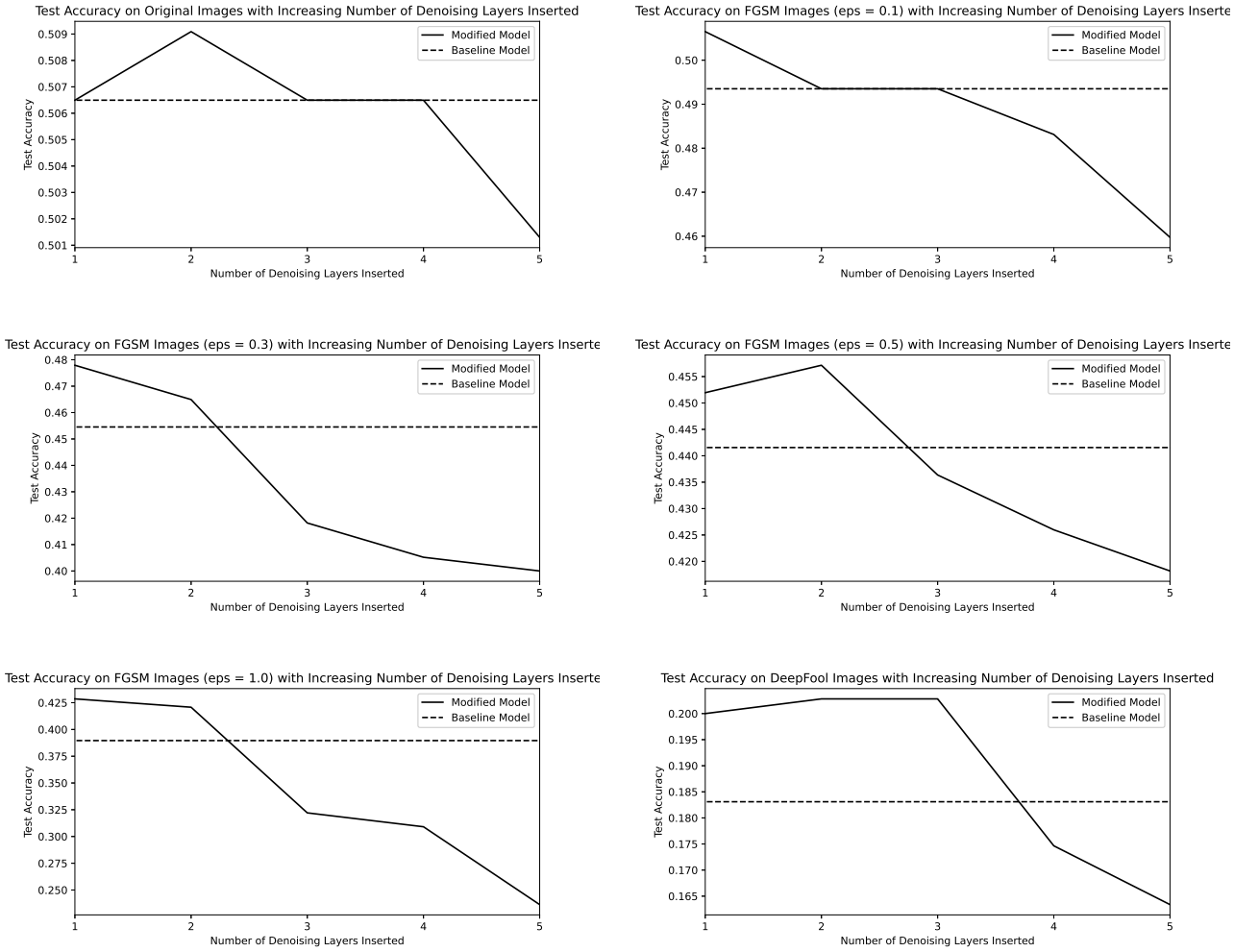



Figure 5: Multiple Layer Insertion Experiments