 ISSN 2714-7533

Multi-SAP Adversarial Defense for Deep Neural Networks

Shorya Sharma [a, 1, *]

[a] School of Electrical Sciences, Indian Institute of Technology Bhubaneswar, India
[1] ss118@iitbbs.ac.in
* corresponding author

ARTICLE INFO

Article history
Received December 10, 2021
Revised December 27, 2021
Accepted Mar 19, 2022

Keywords
Adversarial attacks
Adversarial defenses
Stochastic activation pruning
Deep neural networks
PGD
FGSM

ABSTRACT

Deep learning models have gained immense popularity for machine learning tasks such as image classification and natural language processing due to their high expressibility. However, they are vulnerable to adversarial samples - perturbed samples that are imperceptible to a human, but can cause the deep learning model to give incorrect predictions with a high confidence. This limitation has been a major deterrent in the deployment of deep learning algorithms in production, specifically in security critical systems. In this project, we followed a game theoretic approach to implement a novel defense strategy, that combines multiple Stochastic Activation Pruning with adversarial training. Our defense accuracy outperforms that of PGD adversarial training, which is known to be the one of the best defenses against several L_∞ attacks, by about 6-7%. We are hopeful that our defense strategy can withstand strong attacks leading to more robust deep neural network models.

1. Introduction

Adversaries can easily fool state-of-the-art deep learning models by making small perturbations in the training samples that are imperceptible to a human. Such perturbed samples are called adversarial samples, and attacks on deep learning models that make use of adversarial samples are called adversarial attacks. For a successful attack, we require adversarial samples to be as close as possible to the original (benign) sample, under specific distance metrics. Therefore, these adversarial perturbations can be found by minimizing an objective function, a distance metric between adversarial and benign samples, subject to certain constraints. The threat model for adversarial attacks can be one out of black-box, grey-box or white-box, depending on the attacker's knowledge of the target deep learning model. We plan to develop our defense in a black-box setting, where we assume that the adversary has no knowledge about the target model.

In an adversarial setting, "attacker" and "defender" can be thought of as two parties playing a multi- move game, taking alternative turns to play their move. A successful "attack" will be a gain for the attacker and a loss for the defender, and likewise, a successful "defense" will be a gain for the defender and a loss for the attacker. Thus, this approach manifests itself into a zero-sum two-person

game. We hypothesize that such a zero-sum game approach will enable us to develop a strong adversarial defense.

We designed our mixed strategy defense that builds on the Stochastic Activation Pruning (SAP) algorithm. Stochastic Activation Pruning is a method which stochastically drops out nodes in each layer during forward propagation. The SAP function converts the activation map of a layer to a multinomial distribution and samples the nodes with replacement with a probability proportional to their activation magnitudes. Unlike the original SAP algorithm [15], we leverage the SAP method to create a novel technique utilizing multiple SAP (Multi-SAP) networks. Instead of creating a single model, this multi-SAP method iterates over the SAP function multiple times creating differently pruned networks attributed to the variation in sampling in every iteration. We implemented two defense strategies using multi-SAP on top of adversarial training which improved PGD adversarial defense accuracy by roughly 6-7%.

2. Related Works

There has been an extensive repertoire of work on adversarial attacks and defenses, a few of which have been discussed in the sections below.

2.1 State-of-the-art Attacks

As discussed in Section 1, adversaries can fool deep learning models by making small perturbations in the training samples, that are imperceptible to the human eye. This creates adversarial samples, and such attacks against deep learning models are called adversarial attacks. For a successful attack, we require such adversarial samples to be as close as possible to the original (benign) sample, under specific distance metrics (of which the most commonly used metric is the L_p metric). Therefore, these adversarial perturbations can be found by minimizing an objective function, a distance metric between adversarial and benign samples, subject to certain constraints.

For adversarial attacks on image classification, one of the most common distance metrics has been the L_p norm [1]. For instance, the Limited-Memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method minimizes the L_p norm between the benign and adversarial sample, subject to the constraint that their labels are not equal [2]. Goodfellow et al proposed an untargeted attack (Fast Gradient Sign Method (FGSM)), that generates samples in the L_∞ neighborhood of benign samples. They performed one-step updates along the direction of the gradient to increase the loss in the steepest direction [3]. Following this work, Kurakin et al implemented a Basic Iterative Method (BIM) that improved the performance of FGSM by running a finer iterative optimizer [4]. Building on this, Dong et al incorporated momentum memory into BIM, called the momentum-iterative FGSM (MI-FGSM), inspired by momentum-based optimization [5]. However, the most successful attacks using this approach have been proposed by Carlini and Wagner, in which they generated L_0, L_2, and L_∞ norm measured samples, and solved an optimization problem similar to the L-BFGS attack. They achieved a 100% attack success rate on the MNIST, CIFAR-10 and Imagenet datasets [6].

Instead of optimizing on individual samples like the techniques discussed above, Zheng et al proposed an attack that optimized over the potential adversarial distributions (called distributionally adversarial attack (DAA)) [7]. Another interesting approach is the Universal Adversarial Attack [8], which tries to find the minimum additional perturbations that can be added to a benign sample to make it adversarial. This approach relies on an iterative procedure to update the perturbation vector and has demonstrated successful results against CaffeNet, ResNet, GoogleNet and VGG. Recently, Generative Adversarial Networks (GAN)-based attacks were proposed to generate adversarial samples. In these attacks, Xiao et al trained a generator to maximize adversarial loss and GAN loss to learn the adversarial distribution [9]. Additionally, adversarial patches can be added to specific

sections of a benign sample, that can perturb a patch of the image leading to misclassification [10, 11, 12].

2.2 State-of-the-art Defenses

Despite their breakthrough applications, DNNs are vulnerable to adversarial attacks. Various classes of methods can handle these attacks, namely, adversarial training, randomization, denoising etc. The most common defense technique is adversarial training where adversarial samples generated from adversarial attacks are fed into DNNs during training to increase their robustness against adversaries.

It is similar to a min-max game where we try to minimize the loss from the most effective adversarial samples. FGSM adversarial training uses both benign and FGSM generated adversarial samples to train the DNN. Projected Gradient Descent (PGD) adversarial training has proven to be the best defense against L_∞ based attacks [13]. However, it requires high computational resources making it inefficient. Ensemble adversarial training uses a more diverse set of samples for training making it more effective. Additionally, Generative adversarial training-based defenses employ a generator, a discriminator, and an auxiliary classifier to make the DNN more robust to these attacks.

Further, another class of defense techniques use randomization schemes to mitigate the effect of adversarial perturbations which usually work well for black-box and grey-box attacks but fail for the white-box setting [14]. The random input transformation technique resorts to resizing/padding of input images followed by training. Random noising, aka random self-ensemble, adds a noise layer before each convolution layer and ensembles the prediction results over these random noises to stabilize DNNs. Moreover, stochastic activation pruning is a mixed defense strategy where a random subset of activations is pruned and the survivors are scaled up for compensation [15].

Apart from these techniques, denoising techniques aim at removing the perturbations from the input itself or alleviate their effects on the high-level features learnt by DNNs. Conventional input rectification is one such technique which uses bit reduction and image blurring to remove the perturbations [16]. Defense GAN searches for an image close to the adversarial input to learn the distribution and generate benign samples that can be fed into the classifier. MagNet, an autoencoder based denoising has a detector and a reformer where the detector distinguishes between the adversarial and benign samples and the reformer tries to rectify the adversarial samples into benign samples. Feature denoising tries to minimise a feature level loss function to reduce the difference between the benign and the adversarial samples [17].

2.3 Our attacks and defenses

Gradient based attacks leverage the backpropagation process to develop a perturbation vector for the input image by making a slight change to the input gradients. These methods consider model parameters to be constant and the input to be a variable which in turn is used to obtain the perturbation vector. This vector satisfies the condition of being very similar to the input while fooling the neural network model at the same time such that it's imperceptible to the human eye. FGSM and PGD are two such gradient based attacks.

The two main approaches to perform gradient-based attacks are one-shot and iterative attacks. In one-shot attacks, the attacker takes a single step along the direction of the gradient, whereas in iterative attacks, the attacker takes several steps, instead of a single one. The FGSM method is a one-shot attack, whereas PGD is an iterative attack.

2.3.1 FGSM Attack

The Fast Gradient Sign Method (FGSM) is a form of untargeted attack, proposed by Goodfellow et al, that generates adversarial

$$x' = x + \epsilon.sign[\nabla_x J(\theta, x, y)] \quad (1)$$

samples in the L_∞ neighborhood of benign samples. It generates adversarial images by adding pixel-wide perturbations along the direction of the gradient. An FGSM-generated adversarial sample can be defined as,

In the above equation, x is the benign sample that has been perturbed by the addition of the gradient to obtain an adversarial image, x'. The gradient can be computed as shown in equation (2), where δ is 0 in the first time-step. δ can be adjusted according to a step size α (given by equation (3)), such that δ lies in the range [$-\epsilon$, ϵ], thereby satisfying the inequality in equation (4).

$$g = \nabla_\delta l(h_\theta(x+\delta), y) \tag{2}$$

$$\delta = \delta + \alpha g \tag{3}$$

$$l_\infty norm, \|\delta\|_\infty \leq \epsilon \tag{4}$$

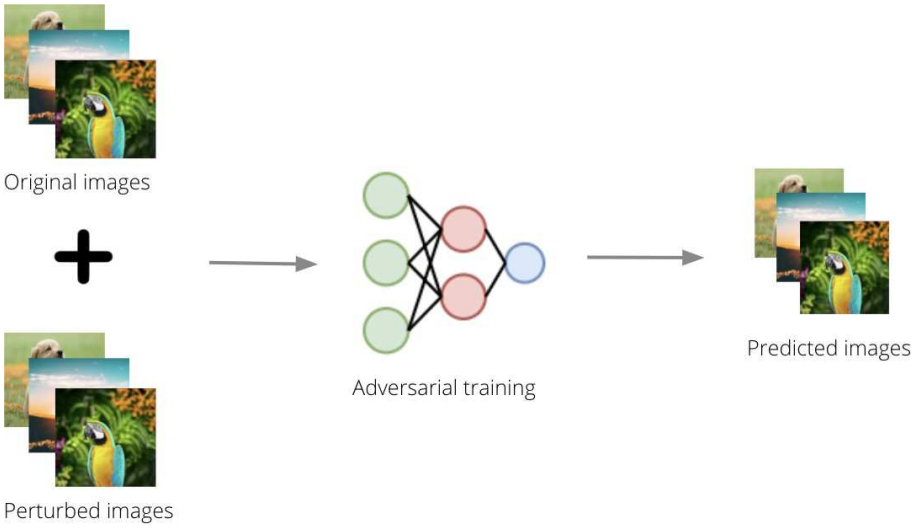

Figure 1. Adversarial training defense illustration

2.3.2 PGD Attack

Projected Gradient Descent (PGD) is an iterative attack where it takes the same step as an FGSM multiple times in the direction of maximising loss on a particular input while keeping the size of perturbation smaller than epsilon. In PGD, one starts from a random perturbation in the L_∞ space around an input and takes a gradient step in the direction of greatest loss in an iterative manner until convergence. To satisfy the constraint, the PGD projects the adversarial samples learned from each iteration in the epsilon constrained L_∞ space of the benign input sample.

$$x'_{t+1} = Projected\{x'_t + \alpha.sign[\nabla_x J(\theta, x'_t, y)]\} \tag{5}$$

2.3.3 Adversarial training defense

Adversarial training is the most common defense technique is adversarial defense where adversarial samples generated from adversarial attacks are fed into DNNs during training to increase their robustness against adversaries. In simple terms, it is like an added data augmentation step where perturbed images are fed into the model as a preprocessing step where these perturbed images are created to best fool the deep neural-network model and train to reduce overfitting. Figure 1 demonstrates a visualization of adversarial training defense.

2.3.4 Stochastic Activation Pruning (SAP)

Stochastic Activation Pruning is a method which stochastically drops out nodes in each layer during forward propagation. The probability of retaining a node is directly proportional to the magnitude of its activation. So, the SAP function converts the activation map of a layer to a multinomial distribution and samples the nodes with replacement with a probability proportional to their activation magnitudes. Equation 6 shows the probability of sampling the j'th activation value, $(h^i)_j$. After pruning, the remaining surviving nodes are scaled up by the inverse of the probability of sampling the nodes over all the draws, following equation 7.

$$p_j{}^i = \frac{|(h^i)_j|}{\sum_{k=1}^{a^i} |(h^i)_k|} \tag{6}$$

$$(m_p{}^i)_j = \frac{I((h^i)_j)}{1-(1-p_j{}^i)^{r_p{}^i}} \tag{7}$$

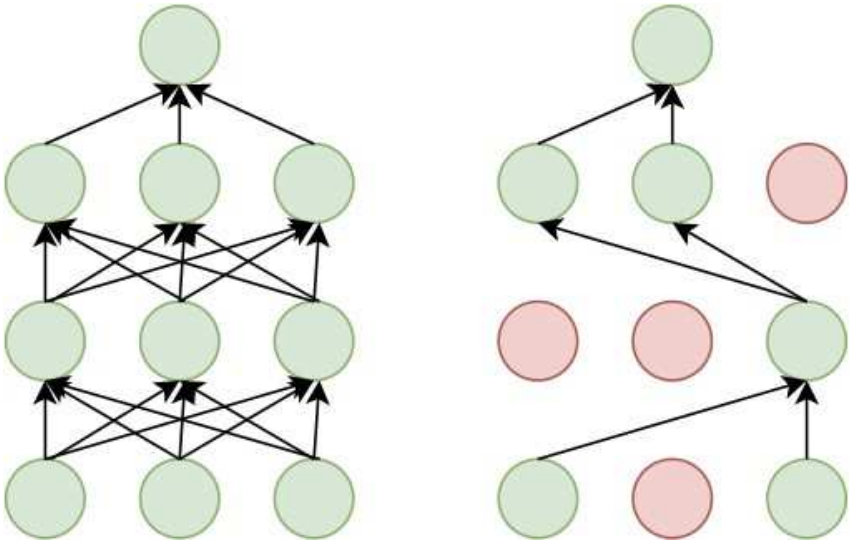

Figure 2. Illustration of SAP

3. Dataset Description

CIFAR-10 is a benchmark dataset used to evaluate image recognition classification tasks. It consists of 10 classes with 6,000 32x32 colour images each. The 10 different classes represent airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships and trucks. The dataset comprises a total of 50,000 training images and 10,000 test images. We have imported the train and test data from torchvision and have also augmented the train dataset by horizontally flipping the images. Since most of the research has been done using the CIFAR-10 dataset, we believe that it will be a good benchmark to compare our model against state-of-the-art attacks and defenses. Another advantage of this dataset is that it is computationally modest.

4. Evaluation Metrics

The objective of adversarial attacks is to fool a classifier into misclassifying samples that are only slightly perturbed from original samples in a way that is imperceptible to a human. Accordingly, to evaluate our adversarial attack or defense, we need a metric that can capture how many samples are correctly classified by our classifier in different scenarios such as when the input images are unperturbed, when they are perturbed using a certain attack technique and when the model has been

defended using a certain defense technique. Thus, we use accuracy as our evaluation metric, which we define as the percentage of images that are correctly classified by our classifier. A higher accuracy means that our classifier is doing a good job, whereas a low accuracy means that our classifier is not being able to perform the classification task properly. A good attack would ideally be successful in lowering the baseline accuracy of the classifier, whereas a strong defense would be able to sustain the attack and still give a high accuracy for classification.

5. Description of Baseline Model

Our adversarial attack and defense setting is on an image classification task where we perturb benign input images to confuse a trained deep neural network. For our project, we have used two kinds of convolutional neural networks, namely, ResNet18 and VGG16, as the defender model and the attacker model respectively. The reason behind choosing two different models for attack and defense has been explained in the subsequent sections. We first implemented the CIFAR-10 classifiers, then implemented our attacks on the defense model using the attack model, following which we developed and evaluated our defenses on the defense model. We discuss each of the above steps in detail below.

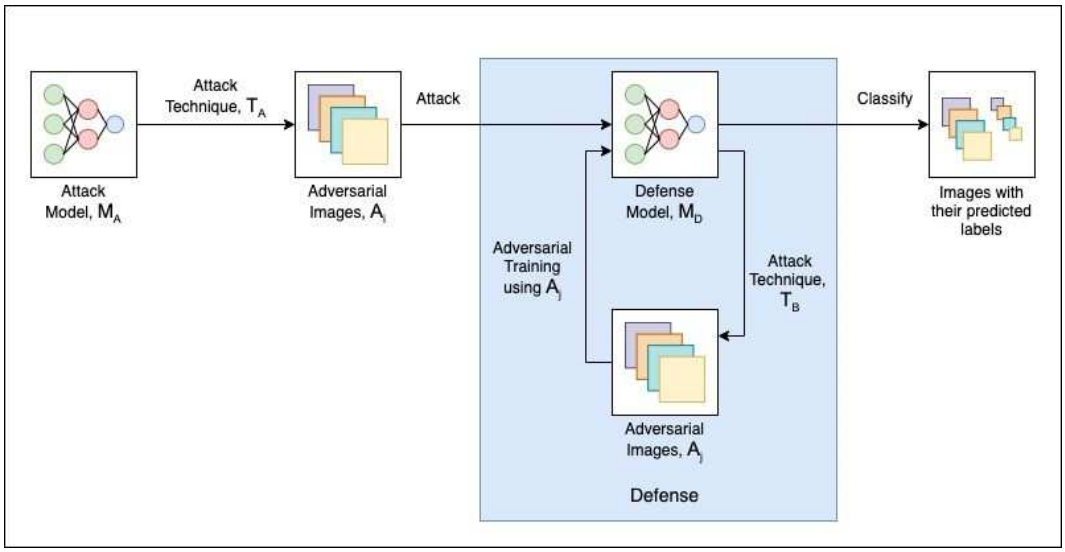

Figure 3. Flow of an Adversarial Attack

5.1 CIFAR-10 Classifiers

5.1.1 Defense model

ResNet18 [20] is a convolutional neural network that consists of 18 layers. The input image for a ResNet model is 224x224, whereas the CIFAR10 dataset consists of images of size 32x32. The parameters of the network have been adjusted to accommodate for the smaller size of images in this dataset. The network consists of several convolutional layers with feature map dimensions of [64, 128, 256, 512], kernel size of 3x3 and appropriate stride and padding values at each layer to obtain the suitable input sizes for each hidden layer. Each convolution layer is also followed by a batchnorm and activation layer, with appropriate skip connections. It is concluded with an adaptive average pooling layer and a 10-way fully-connected linear layer, to obtain the classification results. Xavier normal initialization was used for the weights of the convolutional and linear layers. When this model was trained using an SGD optimizer with a learning rate of 0.15, weight decay of 1e-3, momentum of 0.9, a StepLR scheduler with gamma of 0.85 for 40 epochs, a test accuracy of 92% was achieved.

5.1.2 Attack model

The VGG16 model [21] is also a convolutional neural network, that consists of 16 layers. The parameters of the model are set according to an input image size of 224x224, which have been adjusted to accommodate for the smaller size of images (32x32) in the CIFAR-10 dataset. It consists of several convolutional layers with feature map dimensions of [64, 128, 256, 512], kernel size of 3x3 and appropriate stride and padding values at each layer to obtain the suitable input sizes for each hidden layer. Every convolutional layer is also followed by batchnorm and activation layers, following which every second convolutional layer consists of a max pooling layer, with a kernel size of 2x2 max pooling layer and a 10-way fully-connected linear layer, to obtain the classification results. Xavier normal initialization was used for the weights of the convolutional and linear layers. When this model was trained using an ADAM optimizer with a learning rate of 1e-3, weight decay of 5e-6, momentum of 0.9, a StepLR scheduler with gamma of 0.85 for 40 epochs, a test accuracy of 89.32% was achieved.

Table 1. Classification accuracy of various combinations of attacks and defenses

Attack Model (VGG 16)	Defense Model (ResNet18)			
	Base Classifier	FGSM Trained Model	PGD Trained Model	FGSM & PGD Combined Model
PGD Attack	30,64%	37,55%	61,74%	64,26
FGSM Attack	37,26%	71,91%	71,76%	71,47%

5.2 Attacks

Most of the existing work on adversarial attacks using L_p norm uses L_∞ attacks, and hence, for our baseline attack, we decided to concentrate only on L_∞ attacks. L_∞ norm is defined as the maximum of the absolute values of a vector's components. L_∞ distance or max norm distance looks at the maximum pixel difference between the actual image and the adversarial image. The objective of an L_∞ adversarial attack is to limit the L_∞ norm of the noise to a certain value(epsilon) such that the adversarial image is as close as possible to the input but at the same time different enough for the model to misclassify. We decided to implement two of the most common L_∞ attacks, namely Fast Gradient Sign Method (FGSM) and Projected Gradient Descent (PGD).

We implemented these attacks using the Pytorch framework, following the equations described in Section 2. We followed the approach illustrated in Figure 3 to emulate the attacker. We used the attack model M_A (VGG16) to generate adversarial image set A_i using an attack technique T_A, and then fed these adversarial images into our defense model M_D (ResNet18) for classification. For the attack technique T_A, we experimented with both FGSM and PGD attacks, the choice of hyper-parameters for which are discussed in the Evaluation Section 5.4.

5.3 Defenses

We chose to go ahead with adversarial training for our baseline defense strategy because it is a very common method of adversarial defense, and it is quite trivial to implement. Our defense approach using adversarial training followed the workflow as illustrated in the blue "Defense" box in Figure 3. We used an attack technique T_B on our defense model M_D (ResNet18) to generate adversarial image set A_j, and then trained M_D using images from A_j. In our approach, we experimented with generating A_j using both FGSM and PGD attack methods, and tested our defense against both the attack techniques. We used the same code that we had used for our attack implementations to generate adversarial samples, but for defense, we used the defense model M_D (ResNet18) instead. We discuss the effectiveness of the defense and summarize our findings in the next subsection.

5.4 Experimental Evaluation

We tried to emulate a setting as close to a grey-box setting as possible in our baseline experiments. We first assumed that both the attacker and the defender do not know the kind of model that is being used by the other party. We achieved this by using two different types of models for attack and defense, namely, ResNet-18 as our defense model M_D and VGG16 as our attack model M_A.

We first attacked the defense model M_D with adversarial images generated from the attack model M_A using both FGSM and PGD attacks separately. For the FGSM attack, we tuned the hyperparameter ϵ to obtain the best attack. Using an ϵ value of 0.15, the generated perturbed images were successful in bringing down the accuracy of M_D from 92% to 37.26%. For the PGD attack, we tuned three hyper-parameters (α, ϵ and number of iterations) such that the product of ϵ and number of iterations is approximately equal to the value of α. Using the values of 50, 1/255 and 0.2 for α, number of iterations and ϵ respectively, the generated adversarial images brought down the accuracy of M_D from 92% to 30.64%.

We then implemented our defenses on M_D and evaluated the robustness of the defense against both FGSM and PGD attacks. The hyperparameters that we chose for both FGSM and PGD adversarial training were different from those in our attacks, and this was another attempt to emulate a grey-box

setting - the defender does not know what hyperparameters were chosen by the attacker even if they know the attack method. Our first defense mechanism was training M_D using FGSM generated adversarial images from M_D, for which we chose an ϵ value of 0.05. This defense was able to resist well against FGSM attack, and the model accuracy came down to only 71.91%, in contrast to the initial drop to 37.26%. However, this defense performed poorly against PGD attack, and the accuracy of the M_D still came down to 37.55%.

Our second defense approach involved training M_D using adversarial images generated using the PGD attack on M_D, and we chose the hyperparameter values of 40, 2/255 and 0.3 for α, number of iterations and ϵ respectively. This defense was able to resist FGSM attack almost as well as the FGSM adversarial training approach, and the defended model achieved an accuracy of 71.76%. It performed decently, though not as effectively, on PGD attack too, with M_D achieving an accuracy of 61.74%.

We also tried a combined defense technique, by adversarially training M_D using both FGSM and PGD generated adversarial images, and this performed almost as well as PGD defense against FGSM attack, with M_D achieving an accuracy of 71.47%, but slightly better than PGD defense against PGD attack, enabling M_D to achieve an accuracy of 64.26%. The table in figure 4 gives a summary of our classification accuracy for various attacks and defenses.

6. Our Contribution

6.1 Intuition

A preliminary strategy that we used in our baseline was to have multiple L_∞ attacks on our defense model (ResNet18) and measure the impact of each on the accuracy. Another reason to have multiple attacks was to train our defense model on more than one to make it more robust. Ideally, we would want to have our defense model to be adversarially trained against all possible attacks, but this can be a computational nightmare. Since we can't adversarially train our defense model on all possible attacks, we hypothesize that having a defense network consisting of multiple adversarially trained models that are different from each other, and taking a combination or intelligent selection of the classification results from these diverse models is more robust against a variety of attacks. This method should make it difficult for the attacker to emulate the defense network since implementing an attack against a large number of models is computationally expensive.

We can also tie this strategy back to the zero-sum two player game in game theory, in which the maxmin value is the highest value that the attacker (Player 1) can be sure to get without knowing the defender's (Player 2's) moves, and equivalently, it is also the lowest value the defender can force the attacker to receive when they know their moves. Thus, we can replicate this game theory based mixed strategy by using a set of diverse models to reduce the reward that an attacker gets by making an intelligent classification decision that is based on the input, that is, the attacker's move.

6.2 Diverse Network Creation

To test our hypothesis, our first step was to create a set of multiple models. We used three different methods to create different models from the ResNet18 base network. The first method was to use different subsets of the training dataset to train our model while keeping the accuracy levels close to the baseline model. The second method was to use different weight initializations and train the model on the entire training dataset. The final method involved the addition of skip connections every fourth and sixth layers in the ResNet architecture.

The second step was to adversarially train these multiple models because we wanted to leverage both diversity and adversarial training to defend against an attack. Diversity among the models would ensure that not all the models fail against a particular attack and that the majority of them will still succeed against the attack. This way we can take the majority vote from these models and predict the correct class. Thus if not all, some of the robust networks will be able to defend against the attack. We created a pool of models containing both vanilla and adversarially trained models. However, having multiple models was not enough and it was important to ensure that these models were diverse.

For checking the diversity of the models in the pool that we created, we tried to find the correlation between the models. A lower correlation between two models would indicate that they are more diverse, whereas a higher correlation would indicate that they are less diverse. We computed the correlation between two models by calculating the cosine similarity between the gradients of the loss of the models with respect to the test datum . The pairwise correlation values that we obtained using this method for our set of models were quite low and seemed to suggest that our models were diverse. However, when we tested these models against FGSM attack, they performed as poorly as our baseline model, and in some cases even worse. Given our hypothesis that having a diverse set of uncorrelated models implies that not all models fail against a particular attack, we also tried to randomly pick a model from our pool to classify each input, hoping that the inherent diversity would yield more robustness. However, we found that this technique did not work either, and performed poorly against the attack.

We thus realized that the low values of correlation as a test of diversity was misleading. As an alternative method, we turned to a naive method of calculating the number of agreements and disagreements in the predictions of the models to gauge how diverse they are from one another. If a set of models are robust to an adversarial image, we would expect them to give the correct prediction, and be in agreement with each other. However, if the attack is successful, then the models would give incorrect predictions. In this scenario, if we look at the incorrect predictions of the set of models and find that they are the same incorrect predictions, then we can say that the models are not diverse from one another. Our calculated agreement and disagreement numbers between the models indicated that they were not very diverse to one another, but it was still difficult to ascertain anything about the diversity using these numbers. We summarized all of these experiments and the results in Section 6.4.

6.3 Stochastic Activation Pruning

After conducting the diversity tests as explained in the above section, we realized that our approaches for creating models were not producing a very diverse pool of models. Thus, we tried to find a new method to create a pool of diverse networks. We decided to try Stochastic Activation Pruning (SAP)[15] which is activated after receiving the input (attacker's last move). This was a better approach because instead of finding new methods to create new models and training them, we could just use our existing models and prune them without any need for further training, saving us tons of time.

Stochastic Activation Pruning is a method which stochastically drops out nodes in each layer during forward propagation. The probability of retaining a node is directly proportional to the magnitude of its activation. So, the SAP function converts the activation map of a layer to a multinomial distribution and samples the nodes with replacement with a probability proportional to their activation magnitudes. After pruning, the remaining surviving nodes are scaled up by the inverse of the probability of sampling the nodes over all the draws. In our case, we are performing SAP after every ReLU layer and sampling 100% of the nodes every layer. Unlike the SAP paper, we wanted to leverage the novel SAP method to create multiple SAP (Multi-SAP) networks instead of creating just one by iterating over the SAP function multiple times which creates differently pruned networks attributed to the variation in sampling in every iteration.

6.3.1 Novel Multi-SAP Defense Approach

Our updated defense strategy was to apply Multi-SAP on adversarially trained models and then take a majority vote of classification results from these models. We experimented with two defense strategies to evaluate the efficacy of adding Multi-SAP on top of our adversarially trained models. In the first approach we built a set of 50 networks by applying SAP with 50 iterations to one adversarially trained base network. The second approach was to take a pool (for example 5) of adversarially trained networks and apply SAP on all of them for 10 iterations, each giving us 50 networks. This combined with the initial five adversarially trained models gave us a total of 55 models. Both the strategies were followed by taking a majority vote of classification results obtained from the diverse set of networks to obtain a final prediction. A visualization of this can be observed in Figure 4.

6.4 Experimental Evaluation and Results

6.4.1 Creation of multiple networks

Our first step towards testing our hypothesis was the creation of multiple models from the ResNet18 base network. While creating these multiple models, we ensured that the classification accuracy was close to that of the baseline model (92%). Several methods were implemented:

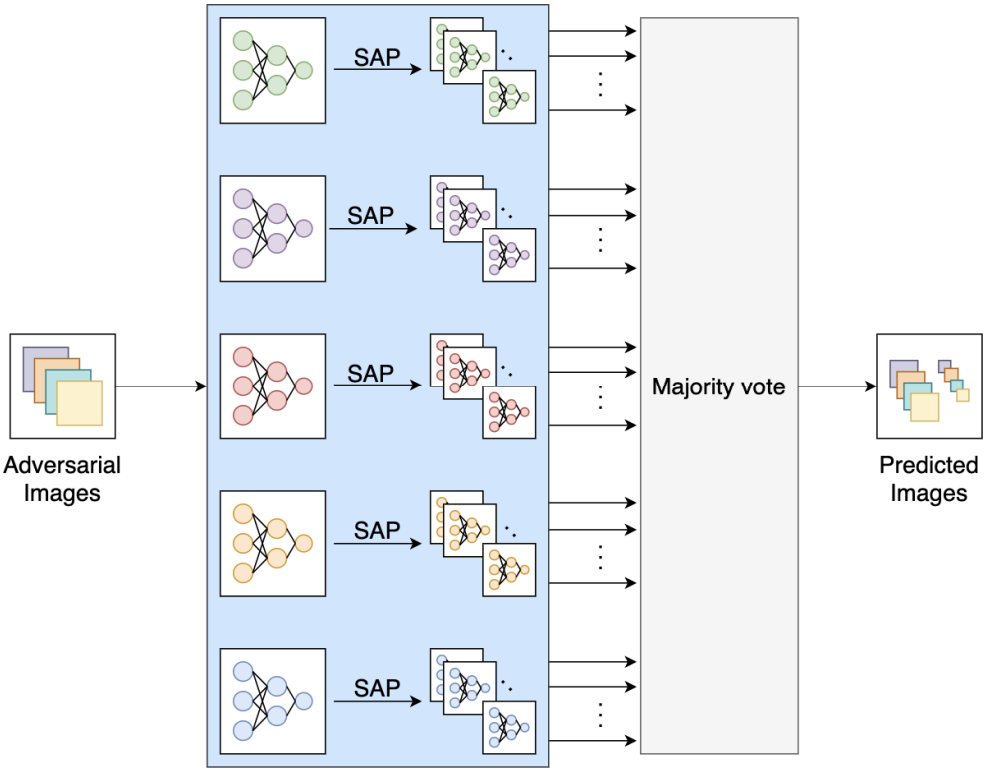

Figure 4. Visualization of our novel Multi-SAP approach

1. **Different training data subsets and random seeds:** We took different subsets of the training dataset and trained them separately to obtain multiple models. We considered a range of values between 30% to 100% of the training data. However, it was observed that considering less than 50% of the training data brought the classification accuracy to lesser than 80%. Therefore, we retained only those models that were trained on greater than 50% of the training dataset. We created four such models that were generated from 50%, 62.5%, 75% and 87.5% of the training data. We also used different random seeds that resulted in sampling different training data points when 75% of the training data was used.

2. **Different weight initializations:** We used different weight initialization methods for the convolutional and final linear layer, to create different models. For the purpose of our experiments, we have used kaiming-uniform, kaiming-normal and orthogonal weight initializations and trained these on the complete training dataset.

3. **Addition of skip-connections:** Our final method involved the addition of skip connections every 4th and 6th layer. We created multiple models by adding more skip connections, in addition to those that are already present in the ResNet18 model. We created two such models by adding an additional skip connection every fourth and sixth layer.

6.4.2 Diversity tests

Our next step was to check the amount of diversity between the above-created models and if these models are diverse enough to defend against any attack. We did this through the following experiments:

1. **Correlation between gradients:** We measured the correlation using Pytorch's inbuilt function, nn. CosineSimilarity, that measures the cosine similarity between two tensors. We used this function to compute the cosine similarity between the gradients of pairwise models with respect

to the test datum as shown in table in Figure 6. A lower cosine similarity value denotes a lower correlation between the gradients, and therefore, greater diversity between the models.

Table 2. Correlation between gradients

	½ train	5/8 train	7/8 train	¾ train	Orthogonal	Kaiming uniform	Kaiming normal
½ train	-	0,2141	0,1724	0,1464	0,1235	0,1077	0,0882
5/8 train	-	-	0,1993	0,1738	0,1720	0,1381	0,1368
7/8 train	-	-	-	0,0950	0,2074	0,1479	0,1096
¾ train	-	-	-	-	0,2036	0,1990	0,0399
Orthogonal	-	-	-	-	-	0,2167	0,0696
Kaiming uniform	-	-	-	-	-	-	0,1072

From table in Figure 2, we observe that all models have really low correlation values ranging from 0.04 to 0.21 when calculated pairwise. The lowest being between 3/4th train and kaiming-normal models. A low pairwise correlation between the models, from the table above, seemed to imply that our models are diverse. We then carried out the following defense approaches -

(a) We performed the attack against this defense network (consisting of diverse models). Given our hypothesis that having a diverse set of uncorrelated models implies that not all models fail against a particular attack, we tried to randomly pick a model from our pool to classify each input, hoping that the inherent diversity would yield more robustness. However, we found that this technique did not work, and performed poorly against the FGSM attack.

(b) Due to the failure of this defense strategy, we adversarially trained the pool of models using PGD adversarial training and created a new set with the adversarially trained models instead of the vanilla models. Our intuition behind adversarially training these models was that creating more robust individual models might improve our defense. Following the same hypothesis as stated above, we randomly picked a model from the new pool of adversarially trained models and tried to classify each input. However, this technique performed only as well as the baseline single adversarially trained ResNet18 model.

(c) Finally, we tried to obtain a majority vote of predictions obtained from the pool of adversarially-trained models, rather than randomly picking a model. This technique gave a 2-3% increase in the accuracy against FGSM attack compared to random selection.

From these experiments, we realized that the low correlation values that we had obtained might not be a true test of the diversity between the models.

2. **Disagreements in model predictions:** Since low values of correlation as a test of diversity was misleading, we tried an alternative approach to test the diversity between the models. As explained in Section 6.2, we calculated the number of disagreements in the predictions of the models, the results of which are tabulated in Tables 3 and 4.

   The number of disagreements are counted out of the predictions made for the 10,000 test images. As we see that ½ train seems to have more disagreements with the other models for both the attacks however, there is no concrete way to conclude that a particular number of disagreements can confirm diversity.

   Therefore, it was difficult to infer the amount of diversity between the models using the numbers shown in Tables 3 and 4.

After conducting the above tests for diversity, we realised that our initial two steps were not producing a very diverse pool of models. Therefore, we tried to implement the Stochastic Activation Pruning (SAP) method to create a pool of diverse networks.

Figure 3. Number of disagreements in model predictions for FGSM attack

FGSM Attack	½ train	¾ train	orthogonal	7/8 train	Kaiming uniform
½ train	-	3361	3286	3599	3405
¾ train	-	-	2743	2934	3161
Orthogonal	-	-	-	2772	2404
Kaiming uniform	-	-	-	-	3196

Figure 4. Number of disagreements in model predictions for PGD attack

FGSM Attack	½ train	¾ train	orthogonal	7/8 train	Kaiming uniform
½ train	-	3603	3658	4220	3650
¾ train	-	-	3247	4040	3472
Orthogonal	-	-	-	3761	2784
Kaiming uniform	-	-	-	-	4005

6.4.3 SAP-based defense

As discussed in the section above, we implemented two defense strategies using SAP. It is a method to stochastically drop nodes in each layer during forward propagation. A major advantage of this approach is that re-training the model is not necessary. SAP is applied to the activations of each layer in the neural network. The general algorithm of SAP is as follows:

1. For each layer, we first normalize the activations to obtain a probability value.
2. This is followed by converting each activation map into a multinomial distribution, such that the probability is directly proportional to the absolute value of the activations.
3. Random samples are then drawn with replacement from the activation map according to this probability distribution. This method makes it easy to determine if an activation would be sampled at all. If it is not sampled, the activation is set to 0.
4. If the activation is sampled, it is re-weighted to maintain the overall dynamic range of the activations.

6.4.4 Defense Strategy 1

In our first defense strategy, we use one base network, here, the base ResNet18 model and adversarially train it. As mentioned in the above Section, different models are produced every time SAP is applied, owing to the variation in sampling at each iteration. Therefore, we applied 50 random seeds and obtained 50 differently pruned models of the base adversarially trained model. We then obtained a majority vote among the classification results from each of these 50 models. The results are further elucidated in the Table 5.

In the table, we demonstrate our defense technique against six L attacks, imported from the torchattacks library in Pytorch. The first column represents the defense performance when the attacks were performed against a single non-adversarially trained model. This is a vanilla model with no defense applied to it. The second column consists of the results when the base model was

adversarially trained and hence the defense performance improved from the previous one. The final column demonstrates the results of Multi-SAP, which are comparable to those of adversarial defense.

Table 5. Performance of defense strategy 1 against different $L\infty$ attacks on the first 1000 test images of the CIFAR10 test data

L_∞ TORCH ATTACK	Single Non adversarially trained model	Single adversarially trained model	Adversarially trained our SAP defense
BIM	46%	55,70%	**56,70%**
RFGSM	33,60%	50,10%	**51,50%**
APGD	41,30%	54,60%	**55,00%**
TPGD	36%	53,80%	**54,60%**
FFGSM	37,90%	54,80%	**55,13%**
MI-FGSM	41,60%	55,20%	**56%**

Table 6. Performance of defense strategy 2 against different $L\infty$ attacks on the first 1000 test images of the CIFAR10 test data

L_∞ TORCH ATTACK	Single Non adversarially trained model	Single adversarially trained model	Adversarially trained our SAP defense
BIM	46%	55,70%	**62,40%**
RFGSM	33,60%	50,10%	**56,10%**
APGD	41,30%	54,60%	**60,70%**
TPGD	36%	53,80%	**61,20%**
FFGSM	37,90%	54,80%	**60,60%**
MI-FGSM	41,60%	55,20%	**61,30%**

6.4.5 Defense Strategy 2

In our second defense strategy, we utilize 5 different base networks that are obtained using different splits of training data and weight initializations. These 5 networks are then adversarially trained. Now, using 10 random seeds, we create 10 diverse networks from each of these five adversarially trained networks, thereby producing 50 diverse networks. These combined with the 5 non-SAP base adversarial networks produced a total of 55 networks. The results are further demonstrated in the table 6.

In the table, we demonstrate our defense technique against six $L\infty$ attacks, imported from the torchattacks library in Pytorch. The table follows the same format as Table 6 in the previous section. Here, the final column demonstrates the results of this strategy of Multi-SAP, which show an improvement of about 6-7% from the adversarial defense technique. As mentioned in the Literature Review, the PGD adversarially trained defense is one of the strongest defenses against most L attacks [13]. Our defense managed to outperform adversarial training as well, which can prove very promising.

7. Conclution

Adversarial attacks against state-of-the-art deep neural networks is indeed a tough battle to combat, and building models robust to these attacks is imperative for the adoption of such systems in the real world. We looked at this problem from a game theoretic perspective and were successful in designing a defense strategy against $L\infty$ attacks that takes into account the attacker's current move. The results obtained from our novel multi-SAP defense technique demonstrate that the usage of diverse networks on top of adversarial training can prove to be a strong defense. The fact that our defense outperforms PGD adversarial training, one of the most powerful defenses against $L\infty$ attacks, by about 6-7%, is reflective of our contribution to the research in this area.

8. Future Work

While it is impressive to come up with a novel defense strategy that outperforms PGD adversarial training, we think that we can improve our results by intelligently selecting classification results from the diverse models instead of using a simple majority vote. Additionally, detecting whether an input is adversarial or not after the attacker's last move will help us further optimize our defense by running Multi SAP only in the case of an attack. Moreover, we can also tune the sampling parameter in the SAP algorithm and evaluate how it affects the accuracy numbers. We are hopeful that these additional steps will strengthen our defense and help us secure our deep learning systems.

References

[1] Ren, K., Zheng, T., Qin, Z. and Liu, X., 2020. Adversarial attacks and defenses in deep learning. Engineering.

[2] Szegedy C, Zaremba W, Sutskever I, Bruna J, Erhan D, Goodfellow I, et al. Intriguing properties of neural networks. 2013. arXiv:1312.6199.

[3] Goodfellow IJ, Shlens J, Szegedy C. Explaining and harnessing adversarial examples. 2014. arXiv:1412.6572.

[4] Kurakin A, Goodfellow I, Bengio S. Adversarial examples in the physical world. 2016. arXiv:1607.02533.

[5] Dong Y, Liao F, Pang T, Su H, Zhu J, Hu X, et al. Boosting adversarial attacks with momentum. In: Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition; 2018 Jun 18–23; Salt Lake City, UT, USA; 2018. p. 9185–193.

[6] Carlini N, Wagner D. Towards evaluating the robustness of neural networks. In: Proceedings of the 2017 IEEE Symposium on Security and Privacy; 2017 May 22–26; San Jose, CA, USA; 2017. p. 39–57.

[7] Zheng T, Chen C, Ren K. Distributionally adversarial attack. 2018. arXiv:1808.05537.

[8] Moosavi-Dezfooli SM, Fawzi A, Fawzi O, Frossard P. Universal adversarial perturbations. In: Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition; 2017 Jul 21–26; Honolulu, HI, USA; 2017. p. 1765–73.

[9] Xiao C, Li B, Zhu JY, He W, Liu M, Song D. Generating adversarial examples with adversarial networks. 2018. arXiv:1801.02610.

[10] Sharif M, Bhagavatula S, Bauer L, Reiter MK. Accessorize to a crime: real and stealthy attacks on state-of-the-art face recognition. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security; 2016 Oct 24–28; Vienna, Austria; 2016. p. 1528–40.

[11] Parkhi OM, Vedaldi A, Zisserman A. Deep face recognition. In: Proceedings of British Machine Vision Conference; 2017 Sep 7–10; Swansea, UK; 2015.

[12] Brown TB, Mané D, Roy A, Abadi M, Gilmer J. Adversarial patch. 2017. arXiv:1712.09665.

[13] Madry A, Makelov A, Schmidt L, Tsipras D, Vladu A. Towards deep learning models resistant to adversarial attacks. 2017. arXiv: 1706.06083.

[14] Athalye A, Engstrom L, Ilya A, Kwok K. Synthesizing robust adversarial examples. 2017. arXiv:1707.07397.

[15] Dhillon GS, Azizzadenesheli K, Lipton ZC, Bernstein J, Kossaifi J, Khanna A, et al. Stochastic activation pruning for robust adversarial defense. 2018. arXiv: 1803.01442.

[16] Xu W, Evans D, Qi Y. Feature squeezing: detecting adversarial examples in deep neural networks. 2017. arXiv: 1704.01155.

[17] Meng D, Chen H. MagNet: a two-pronged defense against adversarial examples. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security; 2017 Oct 30–Nov 3; New York, NY, USA; 2017. p. 135–47.

[18] Neal RM. Bayesian learning for neural networks. New York: Springer Science & Business Media; 2012.

[19] Yang Z, Li B, Chen PY, Song D. Characterizing audio adversarial examples using temporal dependency. 2018. arXiv:1809.10875.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition. 2015. arXiv:1512.03385

[21] Karen Simonyan,Andrew Zisserman. Very Deep Convolutional Networks for Large Scale Image Recognition.2015.arXiv:1409.1556