

TDK-thesis

Balázs Szolár

Improving Adversarial Robustness of Deep Sparse Neural Networks

EÖTVÖS LORÁND UNIVERSITY

FACULTY OF INFORMATICS

DEPT. OF ARTIFICIAL INTELLIGENCE



Author:

Balázs Szolár

Computer Science MSc

IV. grade

Supervisor:

András Lőrincz

Senior Researcher

Budapest, 2023

Abstract

Deep neural networks (DNNs) have shown exceptional performance in various domains such as image classification, object detection, and natural language processing in recent years. However, DNNs are vulnerable to adversarial attacks, where input data is intentionally manipulated to mislead the models. This causes a significant concern for security-critical systems. Another problem with DNNs is that they lack interpretability, thus they are usually regarded as black-boxes.

Sparse coding offers a solution to both of these problems by having interpretable representations and strong theoretical guarantees for adversarial robustness. Dai et al. achieved a breakthrough with deep sparse neural networks which achieved comparable performance to their nonsparse counterparts. They showed that their proposed architectures, the SDNet and SDNetAll, exhibited increased robustness to input corruptions.

In this work, I have improved Dai et al.'s contribution by enhancing the adversarial robustness and sparsity of SDNet18 on CIFAR10. I have improved its dictionaries, making them less noisy and easier to interpret. Finally, I have applied a novel augmentation method, to maintain similar accuracy in my experiments on CIFAR10 as the baseline SDNet-18.

Contents

1	Introduction	2
2	Theoretical background	5
2.1	Convolutional Sparse Coding	5
2.2	Sparse Dictionary learning Network	6
2.3	Projected Gradient Descent	9
2.4	SaliencyMix	9
2.5	Used metrics	10
3	Experiments	12
3.1	Analyzing the baseline models	13
3.2	Including reconstruction loss	15
3.3	Increasing FISTA iterations	17
3.4	Improving clean accuracy with SaliencyMix	18
3.5	Interpretability of the dictionary	19
4	Conclusion	23
	Acknowledgements	25
	Bibliography	25
	List of Figures	30
	List of Tables	31

Chapter 1

Introduction

Over the past decade, deep learning techniques, including deep neural networks (DNNs), have demonstrated remarkable performance across various domains such as image classification [1, 2], object detection and recognition [3] and natural language processing [4, 5]. Furthermore, the success of deep learning can be attributed to the availability of large amounts of labeled data, powerful computing hardware, and novel optimization algorithms that enable training deep neural networks.

Unfortunately, one critical drawback of DNNs is that they are vulnerable to adversarial attacks [6, 7, 8], which refer to the intentional manipulation of input data to mislead deep learning models. This is a significant concern especially for security-critical IoT systems such as autonomous vehicles (AVs). Even slight perturbations to input data can cause DNNs to make incorrect decisions, potentially resulting in catastrophic consequences. For example, AVs may be fooled by adding an unperceivable perturbation to their input, making the vehicle lose control and potentially causing a traffic accident [9].

There are two types of adversary attacks, depending on the amount of information the attacker has about the model. When the adversary has full information about the model (architecture and weight parameters), it is a white-box attack [10], otherwise it is a black-box attack where the adversary can only make queries and observe outputs [11]. The attack itself is carried out by generating adversarial examples. These are normal inputs, that are modified in such a way that it is indiscernible by a human. In the case of classification, the modifications are such that the adversarial examples are classified by the DNN incorrectly. The generation of adversarial examples has been extensively studied over the past few years, and various attack techniques have been proposed. Some of the most popular methods include fast gradient sign method (FGSM) [6], Iterative FGSM

[12] (also known as projected gradient descent (PGD) in the case when random restarts are used [13]). These techniques have been shown to be effective in generating adversarial examples for a wide range of deep learning models. Although the generation of adversarial examples seems most straightforward for visual data, it can also be applied to other domains, for example natural language processing [14].

Another issue with DNNs is that they are “black boxes”, in a sense that we have no clear theoretical understanding and reasoning for their architectures and hidden representations [15]. The layer operations (such as convolutions in the case of deep convolutional networks), nonlinearity and normalization are all developed with minimizing an end-to-end training loss in mind. Thus, they do not have much data-specific meaning. The result is that the hidden layers and the hidden representations that they generate and the weights that they learn (feature maps and feature detectors in the case of deep convolutional networks) are hard to interpret. The lack of interpretability also contributes to the aforementioned issues with adversarial robustness.

There have been numerous solutions proposed to both of these problems. Against adversarial attacks various defense techniques have been proposed, such as adversarial training, first proposed by Szegedy et al. [16]. The idea is that the model is trained on a mixture of adversarial and clean examples, learning the correct labels for the adversarial examples in the process. The drawback of this method is that performance on the clean data will decrease and the model will be only robust against the particular attack that it was trained on [17]. Other solutions include defensive distillation [18], and gradient masking [19]. Defensive distillation is a technique that adds flexibility to the model’s classification process so the model is less susceptible to exploitation. This is done by training the model on soft targets instead of hard targets. The biggest disadvantage of this method is that given enough computing power and fine-tuning by the attacker, the model can still be reverse-engineered to discover fundamental exploits. Poisoning attack can also affect distillation models, in which erroneous data entries are inserted into the initial training database. Gradient masking involves hiding the model’s gradients during inference to prevent attackers from crafting adversarial examples. One method to do this in the case of classification for example, is modifying the output of the network, such that instead of class probabilities, it outputs a one-hot vector. The point of gradient masking is to make it harder for the adversary to find gradients indicating good candidate directions in which the input can be perturbed. However, even this defense has a flaw: the adversary can train a copy of the original model that tries to replicate the defended model. This copy

model’s gradients can then be used to find adversarial examples that are misclassified by the original model as well.

On the problem of interpretability Chan et al. proposed a white-box network, called ReduNet [20]. They optimize a rate reduction objective, which naturally leads to a multi-layer deep network, that shares common characteristics of modern DNNs, while also having precise optimization, statistical, and geometric interpretation. Unfortunately, the performance of this ReduNet does not reach state-of-the-art results of modern DNNs on modern datasets such as CIFAR10.

A solution to both of these problems is sparse coding. Sparse layers have explicit data modeling, they assume that their input can be represented by a few atoms from a dictionary shared by all data points. This results in interpretable representations, solving the second issue. Sparse coding also has strong theoretical guarantees for handling corrupted data and for adversarial robustness [21, 22]. Until recently, there have been no cases of deep sparse neural networks, that could achieve the performance of their nonsparse counterparts. Dai et al. were the first to make a breakthrough in this problem: they proposed the SDNet and SDNetAll architectures (Sparse Dictionary learning Network) [23]: it has the same architecture as a ResNet [1], with the only difference that for SDNet its first layer and for SDNetAll all of its layers are sparse. A crucial finding was that these models could achieve on par performance with their ResNet counterparts on modern datasets. They also showed that the SDNet and SDNetAll exhibit increased adversarial robustness as well as robustness to input corruptions using the stable recovery property of sparse layers.

In this work, I extended and improved on their results and have made the following contributions:

- I have improved the adversarial robustness and sparsity of SDNet18 on CIFAR10.
- I have improved the dictionary of SDNet18’s sparse layer, making it less noisy.
- I have applied SaliencyMix augmentation, to maintain similar accuracy on CIFAR10 as the baseline SDNet18.

Chapter 2

Theoretical background

2.1 Convolutional Sparse Coding

The Sparse Coding (SC) problem or basis pursuit is the mathematical optimization problem of the form

$$\min_x \|x\|_0 \quad \text{subject to} \quad y = Ax, \quad (2.1)$$

where x is a N -dimensional solution vector (in our case, our output), $\|\cdot\|_0$ denotes the l_0 norm¹, y is a M -dimensional vector of observations (in our case, our input), A is a $M \times N$ transform matrix, also called dictionary (in our case, our weight matrix). If $M < N$ then it is an overcomplete, if $M > N$ then it is an undercomplete dictionary [24]. The rows of the matrix are the atoms and if we assume that our data (y) is generated by the linear combination of only a small number of atoms (this is the sparse model), then we can find its corresponding sparse representation (x) by solving the basis pursuit problem. The result x will be a sparse representation, since we assumed that we only use a small subset of the dictionary's atoms.

Due to this, we are motivated in solving the basis pursuit problem. Because of the l_0 norm this problem is hard to solve, so we can use a relaxation of basis pursuit by using the l_1 norm instead of the l_0 norm. Also, to make the problem unconstrained we remove the requirement of $y = Ax$ and instead aim to minimize the squared error between them. This unconstrained version of the problem is called LASSO [25]:

$$\min_x \frac{1}{2} \|y - Ax\|_2^2 + \lambda \|y\|_1, \quad (2.2)$$

¹Of course, it is not a norm by mathematical definition, this word is only used for simplicity.

with $\lambda > 0$. One frequently used algorithm for the solution of this LASSO problem is the Iterative Shrinking and Thresholding Algorithm (ISTA and its fast version FISTA) [26]. FISTA's iterative update takes the following form by first taking a negative (reconstruction loss) gradient step then applying a proximal operator (denoted as ϕ , in our case it will be a non-negative soft shrinkage operator).

$$x^{[t+1]} = \phi_{\lambda}(x^{[t]} - A^T(Ax^{[t]} - y)) \quad (2.3)$$

Here, the λ is an important hyper-parameter, as we will see in the experiments, since we can control the balance between the sparsity regularization ($\|\cdot\|_1$) and the reconstruction loss ($y - Ax$) with it. This algorithm can be considered as a neural layer, since we are trying to find a (sparse) representation (x , the solution of the LASSO problem) for our input vector y . Of course the construction works the other way too - given a sparse representation and our dictionary we can reconstruct our input. This makes our hidden representation more explainable if our dictionary atoms are explainable too, since the sparse representation is the "selection" of atoms that reconstruct our input in the best possible way.

Convolutional sparse coding (CSC) is an extension of this sparse coding model [27]. The dictionary in this case can be modeled as a banded and circulant matrix, where the band consists of the convolution's kernels in vector shape. Multiplying by this matrix is the same as convolving with the kernels of the convolution. This structure adopted by the convolutional sparse coding model allows the sparsity prior to be applied locally instead of globally: independent patches of y are generated by "local" dictionaries operating over stripes of x . This means that every patch of the input image has a sparse representation with regards to the same local dictionary - this is the main assumption of CSC. This means in theory that an image of a horse could be constructed from the same image patches that an image of a cat would be constructed from.

2.2 Sparse Dictionary learning Network

The previously mentioned CSC layer is an improvement over the regular neural layers in terms of robustness to noise and interpretability that comes from its way of modeling data. However while one layer may be sufficient for simple image datasets such as MNIST, for more complex datasets such as CIFAR-10, CIFAR-100 or ImageNet, incrementally deeper and more elaborate network architectures are needed to maintain top

performance on different tasks.

This problem occurs for both types of networks and is called the *degradation problem*: as we increase the depth of the network, the performance stagnates and eventually even starts to decrease. It has been shown by He, Kaiming, et al. [28] that if we add more layers to a deep enough model then the training error of the model increases. In another paper [1], they verify this finding in their experiments and claim that due to this, the cause of the degradation problem cannot be overfitting. In the same paper, they introduce a new architecture that doesn't suffer from degradation: the *ResNet* (Residual Network). The main idea of the ResNet is including "skip connections" (or "shortcut connections") between certain layers. These skip connections apply identity mapping to the input of a stack of layers and add the result to the output of the stack of layers - thereby allowing the input to "skip" that part of the network. He, Kaiming, et al. propose multiple architectures that use this method, and using them they could solve the degradation problem. They even explore the possibility of 1000 deep neural networks and show that the problem of degradation does not appear even in this exaggerated case.

The *SDNet* and *SDNetAll* architectures (Sparse Dictionary learning Network) proposed by Dai et al. [23] build on the ResNet architecture, but the idea can be generalized beyond it. To solve the issue of degradation for deep sparse neural networks, they use an existing baseline neural network (the ResNet in their case) and they replace their convolutional layers with the aforementioned CSC layer. All other layers such as normalization, nonlinear, and fully connected layers are kept in the model. We can see that this idea is general enough to be applied in further works to state-of-the-art models even in different domains. In this specific case, they use the ResNet-18 and ResNet-34 to produce the SDNet-18 and SDNet-34 - in these models, only the first convolutional layers are replaced with a CSC layer. The other two models are named SDNet-18-All and SDNet-34-All, where all of the convolutional layers are replaced with CSC layers. The architectures of ResNet-18, SDNet-18 and SDNet-18-All can be seen in Figure 2.1.

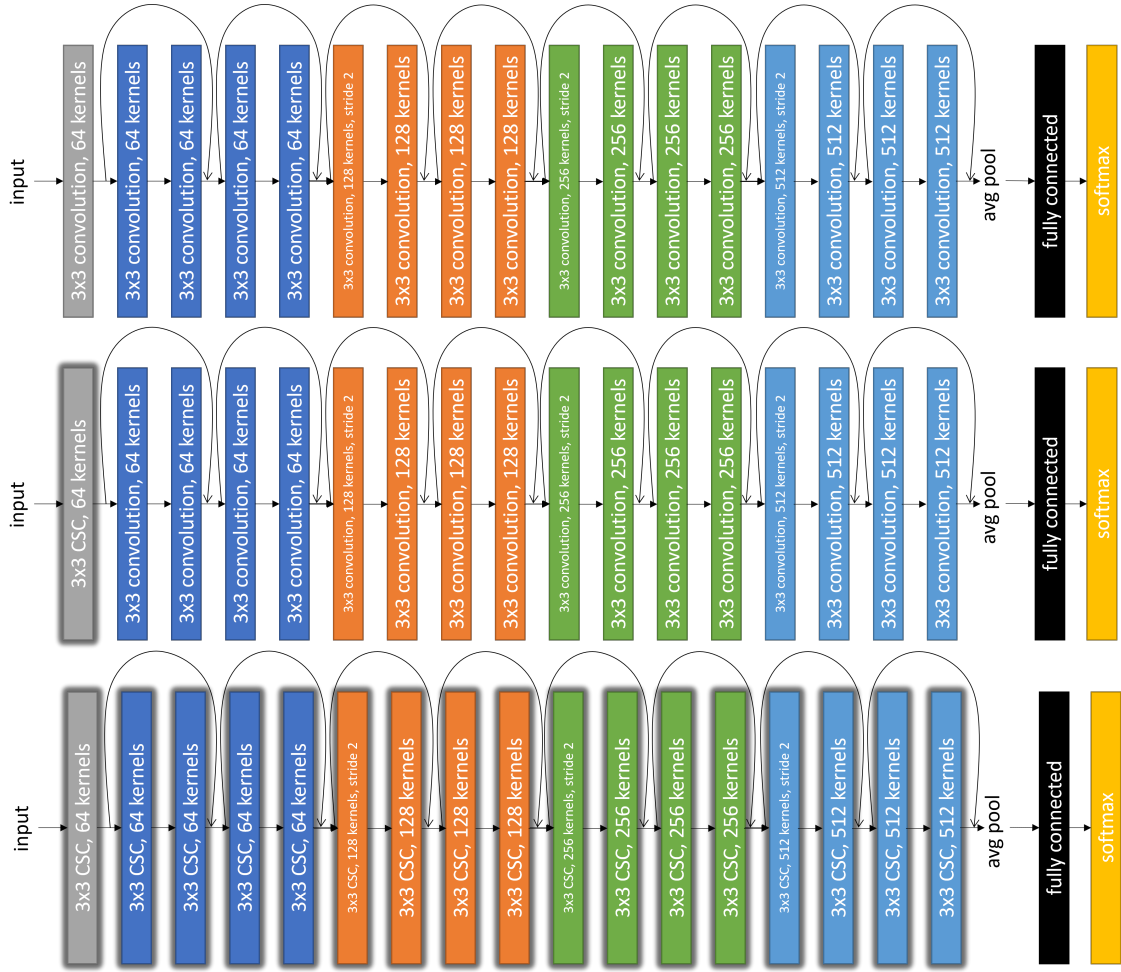


Figure 2.1: The ResNet-18, SDNet-18 and SDNet-18-All architectures (from top to bottom). Colors represent blocks of the network. Every block has 2 skip-connections, which are represented by arrows.

Although the SDNet-18 only has its first layer replaced with a CSC, we will see that even one layer is enough to provide an improvement in adversarial robustness.

Dai et al. show that these SDNet models have equal accuracy on CIFAR-10, CIFAR-100 and ImageNet as the ResNet baseline models. They also show that these models are more robust to input corruptions and adversarial attacks than the baselines. Finally, they utilize these models to visualize the hidden features of the network by reconstructing the images from them. They show that deeper layers can only reconstruct rough contours and suggest that this may be due to the fact that SDNet progressively removes unrelated details. On the other hand, they do not examine the learned dictionaries (kernels) of the network, only show one example, where the learned dictionary is noisy. In Section 3.5 I show that the dictionary can also be improved along with the adversarial robustness.

2.3 Projected Gradient Descent

In this work, I use the same adversarial attack as in the above mentioned SDNet paper: Projected Gradient Descent (PGD). It is a white-box attack, meaning that the weights of the model are available to the attacker. PGD is based on two other methods that I will detail first.

The fast gradient sign method (FGSM) [6] generates adversarial noise using the following form:

$$\varepsilon \text{sign}(\Delta_x J(\theta, x, y)), \quad (2.4)$$

where $J(\theta, x, y)$ is the loss function used for classification (x is the input image, y is the expected output, θ represents the parameters) and $\varepsilon > 0$. This noise is added to the image x and if ε is small enough then the modification will be indiscernible by a human. In practice the gradient required by FGSM can be computed with backpropagation.

An improvement over FGSM is the Iterative FGSM [12]. If we consider the attack as an optimization problem, where we wish to decrease the loss function by modifying the input x then we can see that with FGSM we are practically only taking one gradient step. This is how IFGSM extends FGSM: by introducing an iterative procedure. We wish to get an output that is within an ε sized l_2 or l_∞ norm ball since we want the attack to be indiscernible by a human. To solve this, IFGSM applies per-pixel clipping after each iteration with the $\text{Clip}_{x,\varepsilon}(x')$ function. The result is the following iterative procedure:

$$x_0^{\text{adv}} = x, \quad x_{n+1}^{\text{adv}} = \text{Clip}_{x,\varepsilon}\{x_n^{\text{adv}} + \alpha \text{sign}(\Delta_x J(\theta, x_n^{\text{adv}}, y))\}. \quad (2.5)$$

The hyper-parameters here are the number of steps (maximum number of n), stepsize (α), and ε which decides the size of the attack. We may also choose between using the l_2 or l_∞ norm when clipping the adversarial example.

Projected Gradient Descent (PGD) [13] only differs slightly from IFGSM. While with IFGSM we are initializing with the original image x , with PGD each run starts at a uniformly random point in the l_2 or l_∞ ball around the image. It also utilizes random restarts. In this work I will refer to the PGD's l_2 and l_∞ cases as the L_2 and L_∞ attack respectively.

2.4 SaliencyMix

Several data augmentation strategies exist, that improve the generalization ability of deep learning models. One of the most popular methods is regional dropout [29] that ran-

domly removes regions from the images, forcing the model to focus on less discriminative parts. An improvement of this method is CutMix proposed by Yun et al. [30]. Instead of leaving the removed region in the images blank, it randomly replaces it with a patch from another training image. Then, their labels are mixed according to the ratio of mixed pixels. However, it may happen that the randomly selected patch is from the background of the image, which may be irrelevant to predicting the target objects. In this case, the method would be no better than leaving the chosen patch blank, but since the labels are also mixed, the model will be unable to learn anything from this example.

As a solution Uddin et al. proposed SaliencyMix in 2020 [31]. Instead of choosing the image patches randomly, they use the saliency map of the source image which highlights important objects in it. Then they select a patch where the saliency map is maximal, indicating that the patch contains relevant information. Then they mix this patch with the target image and also mix the labels as in the case of CutMix. They show that this is a more effective data augmentation method. I use this method in Section 3.4 to improve the accuracy of my experiments.

2.5 Used metrics

Sparsity We will define the sparsity in terms of the CSC layer. In Equation 2.2 we can see the two terms we are trying to optimize, the second of which is the sparsity regularization term. As mentioned before, with the λ hyper-parameter (also called bias) we can control the balance between the two terms. If our λ is high, our layer will produce outputs that have a lot of zero entries - in other words, it will produce an activation with a lot of inactive neurons. On the other hand, if our λ is low, we will get a lot of active neurons, making the layer less sparse. We will define the sparsity of a layer by the ratio of inactive neurons in its activation:

$$sparsity(x) = \frac{||x||_0}{n} \quad (2.6)$$

where x is the output of a layer and n is the number of components of x .

Reconstruction loss As mentioned before, CSC layers can reconstruct the input, since they are optimized to produce outputs whose reconstruction is close to the original image - this is the first term of Equation 2.2. Using a sparse representation produced by the CSC layer and its dictionary we can reconstruct the input. This is done by

applying a transposed convolution operator over the representation (also known as deconvolution, for more information see [32]). We define the reconstruction loss as the mean squared error (MSE) between the original and reconstructed input. A few examples of how the quality of the reconstruction corresponds with the value of the reconstruction loss can be seen in the figure below.



Figure 2.2: Examples of different reconstruction loss values. The top two rows of each block are the original images and the bottom two rows are their reconstructed versions. The reconstruction loss of each case is denoted above them.

Robust accuracy Robust accuracy refers to the accuracy of the model when it is provided input modified by the PGD (projected gradient descent) attack (for details, see Section 2.3). $L_2 = 0.5$ and $L_\infty = 8/255$ refer to the L_2 and L_∞ attacks with ϵ value of 0.5 and 8/255 respectively - similarly as in the SDNet paper.

Chapter 3

Experiments

In my work the goal was to improve and extend the work of Dai et al. [23]. First I analyzed their published trained models and considered them as a baseline in Section 3.1. In the following sections - due to resource and time constraints - I only considered SDNet-18 and CIFAR-10. The other models and datasets could be a topic of future work. In Section 3.2 I analyzed the effect of different λ values and improve on the results by training also for reconstruction loss. In Section 3.3 I considered the effect of different number of FISTA iterations. Next, I improved results in the mentioned sections by training with SaliencyMix in Section 3.4. Finally, I presented the effects of training for reconstruction loss on the dictionaries of the models in Section 3.5 and also presented a visualization of the best result's interpretability.

The training parameters for the experiments are the same as in the work of Dai et al.: 220 epochs, 128 batch size, cosine learning rate scheduler with an initial learning rate of 0.1. The optimizer is SGD with a Nesterov momentum of 0.9 and weight decay of 5×10^{-4} . The training loss is cross entropy with added reconstruction loss (detailed in Section 2.5) when specified.

In every case the parameters for the PGD attack are the following: the stepsize for L_2 is 0.0125, for L_∞ it is 4/255. The number of steps in both cases is 20. These were chosen so that the robustness accuracies for the baselines would be close to the reported results in the SDNet paper.

3.1 Analyzing the baseline models

Dai et al. published their classification accuracy results on normal, noisy and adversarial data. However, they did not include an analysis of their SDNet models, which would be crucial for future research to enable comparisons with new models. For this reason - and also to have a baseline for fair comparison -, in this section I provide a deeper analysis of their models trained on CIFAR-10. All metrics here are calculated on the CIFAR-10 test set. Dai et al.'s code and trained model weights are publicly available¹, making reproducing their results an easier task. They provide weights for the SDNet-18, SDNet-18-All and also a third model, the SDNet-18-All's no shortcut version which doesn't contain skip-connections. All were trained with a λ of 0.1.

The most important metric of these models is the top-1 accuracy, since without achieving satisfactory results, any further work would lack any significance or practicality. In Table 3.1 we can see that all sparse models achieve close results to the nonsparse ResNet-18, which was the main breakthrough of their work. The no shortcut version of SDNet-18-All performs worse, which is not surprising, since the skip-connections provide the ability to improve the performance in the ResNet-18 as well. Also, we can see a slight decrease in accuracy from SDNet-18 to SDNet-18-All, since replacing all of the layers with CSC layers makes the training harder: the individual layers are optimizing themselves to create sparse outputs that can reconstruct their inputs.

Regarding robust accuracy, the SDNet-18 result is similar to the one presented in the SDNet paper - there is only a minor increase in robust accuracy. What the original paper didn't include, are the results for SDNet-18-All and its no shortcut version. This is probably due to the fact that they do not provide any more robustness than the SDNet-18 - this is surprising, since we would expect that more CSC layers would provide more defense against adversarial attacks. Since the scope of this work is limited to the SDNet-18, the analysis of this problem is not included.

¹<https://github.com/Delay-Xili/SDNet>

	Top-1 Accuracy	Robust Accuracy ($L_2 = 0.5$)	Robust Accuracy ($L_\infty = 8/255$)
ResNet-18	95.78%	29.28%	0.05%
SDNet-18	95.33%	31.84%	0.11%
SDNet-18-All	95.03%	31.81%	0.23%
SDNet-18-All (no shortcut)	94.89%	31.27%	2.6%

Table 3.1: Accuracy results for the baseline models.

We can conclude that the improvement of these trained SDNet models (our baselines) over the ResNet model in terms of adversarial accuracy is very minor. But an interesting result in the SDNet paper suggests that this can be greatly improved. Using the trained SDNet-18 model (trained with $\lambda = 0.1$, as mentioned before), they claim that tuning the λ hyper-parameter on the CIFAR-10 test set with grid search (after training) from 0.1 to 0.9 with 0.1 intervals can achieve a robust accuracy of 35.18% for L_∞ and 62.8% for L_2 . Unfortunately, Figure 3.1 shows that this result is not reproducible. By increasing λ , the accuracy decreases. It only increases for the L_∞ case because it's value is below 10% - when λ becomes big enough for the layer to be 100% sparse, all accuracies become 10% (completely random). So the L_∞ accuracy increases only because it is converging towards 10%.

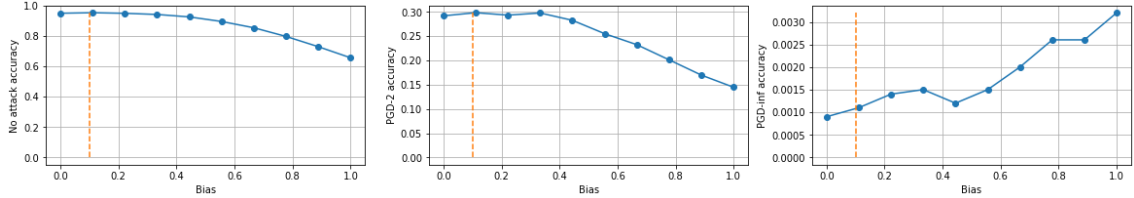


Figure 3.1: Trained SDNet-18’s accuracy, L_2 and L_∞ robust accuracy with different biases (λ values). The orange line indicates the original bias value of 0.1 that was used during training.

Using the definition of sparsity in Section 2.5 we can see the sparsity of each layer in the baseline models in Figure 3.2. All of the SDNet models were trained with a λ of 0.1. We can see clearly that all normal layers (layers 2 to 17 in SDNet-18) have 0% sparsity, meaning their neurons are active for all inputs, while CSC layers are at least 40% sparse in all cases. For SDNet-18-All, the layers sparsities vary between 40% and 80%, despite having the same λ . This is another issue that comes with training deep sparse neural networks - finding the optimal λ value is a difficult process, especially with multiple layers, since the sparsity of deeper layers depends also on the λ values of previous layers.

One interesting solution to this could be k-sparsity [33], where the layer’s activations are made sparse by choosing its k most active neurons, and deactivating the others. This way, a constant sparsity could be retained throughout training, and hyper-parameter tuning would be simplified to finding just k. Since the scope of this work is limited to only improving SDNet-18 - where only one λ needs to be optimized - this approach is not considered.

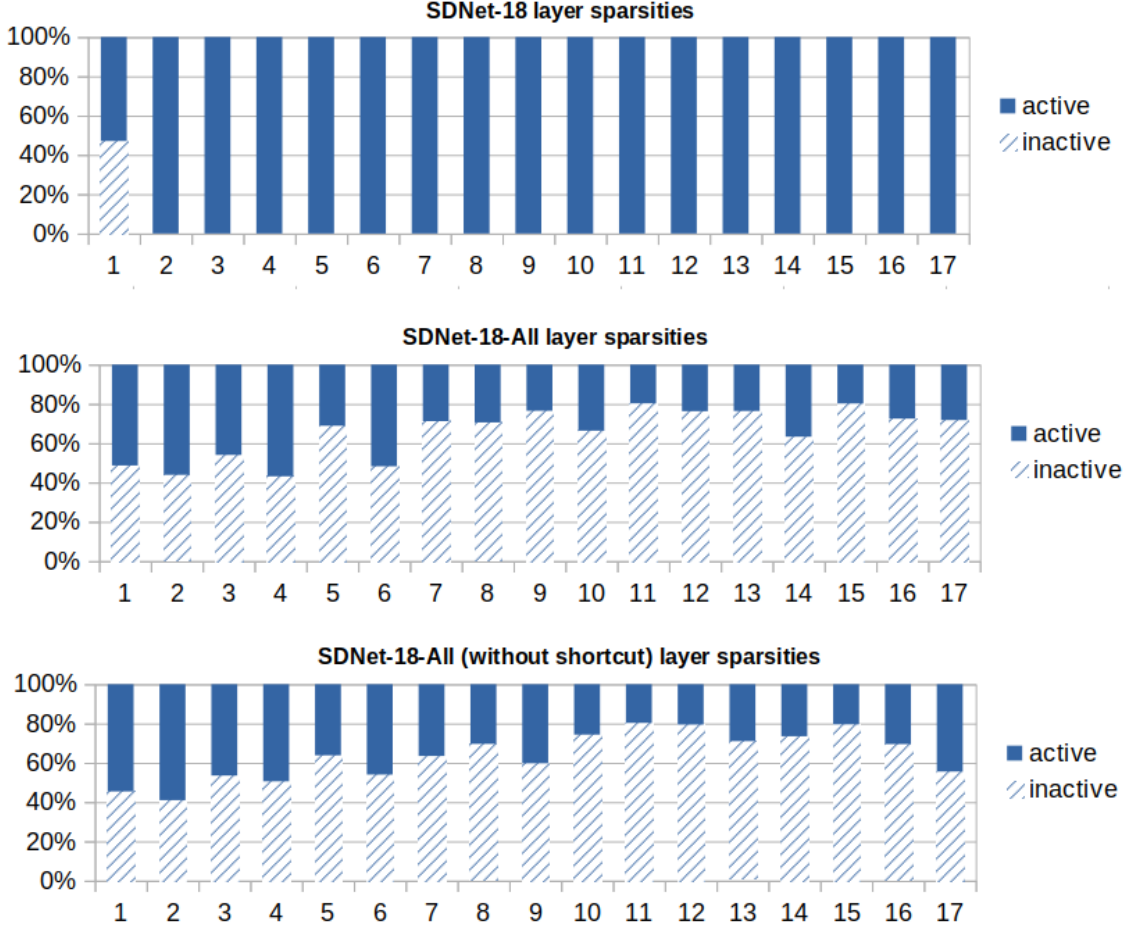


Figure 3.2: The SDNet-18, SDNet-18-All and SDNet-18-All no shortcut model’s layer sparsities. On the x axis are the layer indices. On the y axis is the sparsity of the layer (which is the percentage of inactive units).

3.2 Including reconstruction loss

In this section we will see the effect of training with an added reconstruction loss. First, let us see the effect of different λ values on the training. The following SDNet-18 models were trained with different λ hyper-parameters. As we can see on Figure 3.3 (denoted with "reconstruction coefficient = 0"), the top-1 accuracy is slowly decreasing

as we are increasing the bias. Also, the sparsity and reconstruction loss are increasing, at a faster rate. As our layer is becoming sparser and sparser, it becomes harder for the network to maintain the same accuracy, thus the performance decreases, while the reconstruction loss increases for the same reason. Compared to the SDNet-18 baseline shown in Section 3.1, we can see that sparseness can be increased well above the baseline of 50%, while maintaining the same accuracy. With $\lambda = 2$ for example, the accuracy is still 93.96%, while sparsity is 78%. Unfortunately, reconstruction loss increases at a similar rate as the sparsity. This is due to the λ trade-off mentioned before, between the two terms in Equation 2.2. Higher λ favors higher sparsity, while neglecting the reconstruction loss even more. On the other end, lower λ values result in a non-sparse layer, but low reconstruction loss.

Regarding the robustness, we can see no significant relation between λ and the robust accuracy. As mentioned before, L_∞ accuracy only increases since it is converging towards 10%.

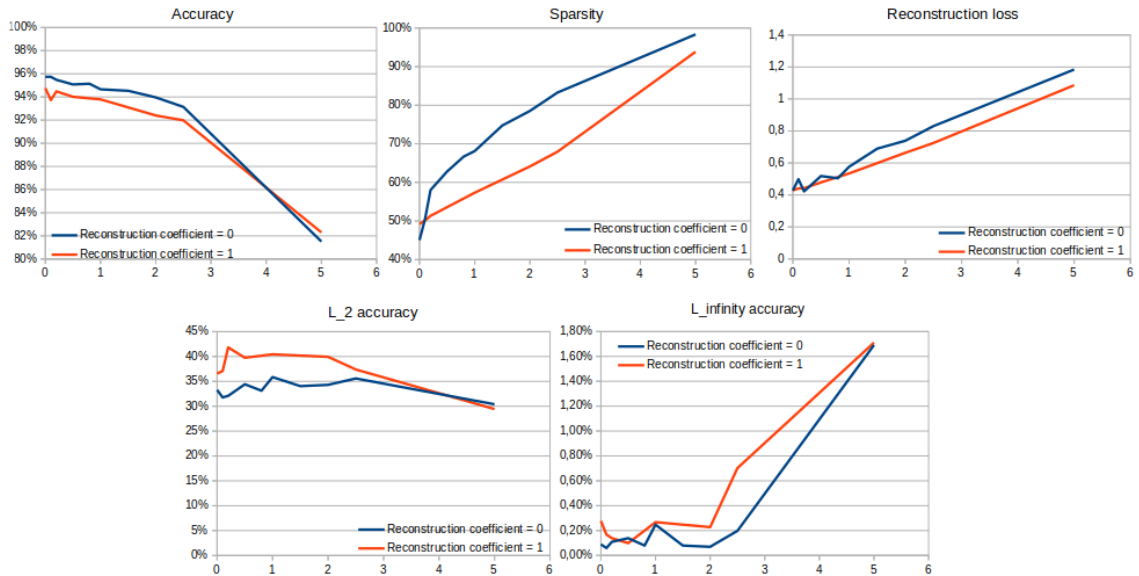


Figure 3.3: The SDNet-18 model’s accuracies, sparsity and reconstruction loss when trained with different λ values (x axis). Sparsity and reconstruction loss corresponds to the first layer of the model. Note that the range of the y axis differs for every metric.

Next, let us see the effect of adding reconstruction loss to the training loss. In Section 2.5 we have seen what values the reconstruction loss takes. They are in the same magnitude as the training loss, so I have used a coefficient of 1 for the reconstruction loss. Again, SDNet-18 models were trained with different λ values, but with this modified loss this time. The results can be seen on the same figure as before. We have slightly

decreased the top-1 accuracy, by around 1% which was expected, since we changed the loss. The reconstruction loss didn't decrease significantly, which could be surprising since we trained on decreasing it. But we need to remember that during training we are modifying the weights/dictionaries, while the reconstruction loss is being optimized by the layer (the FISTA algorithm). The algorithm creates an output that is both sparse and has the best possible reconstruction loss given the dictionary. We will see later that to improve the reconstruction loss, we need to increase the iterations of the FISTA algorithm.

Looking at the robustness results, we can see the advantage of training for reconstruction loss. There is around 5% increase in L_2 accuracy, while L_∞ also slightly improved. For example, with $\lambda = 0.2$ the clean accuracy is 94.48%, while the L_2 accuracy is 41.82%, which is a 10% improvement over the baseline.

3.3 Increasing FISTA iterations

Although the above results are interesting, we are also interested in making the layer sparser, since there are theoretical guarantees on the stability of basis pursuit if the representation is sparse enough [34]. Unfortunately, looking at Figure 3.3 we can see that even though a sparser representation is achievable by larger λ values, it is at the cost of the accuracy and reconstruction loss.

As a solution, we may try increasing the number of FISTA iterations. Dai et al. used only 2 iterations (so far these experiments also used 2), since they have shown that using more iterations doesn't provide significant improvement in accuracy and robust accuracy. But they didn't take sparsity into consideration and had a sparsity of about 50% (Figure 3.2). Using more sparsity (we are aiming for >90%) will provide a basis for basis pursuit's stability guarantees, while increasing the number of iterations will counter the effect of the increased reconstruction loss.

Since increasing the number of iterations also increases sparsity, I chose $\lambda = 2$ as the basis of the following experiments. This case has 78.52% sparsity, which leaves enough room for further sparsity to be accommodated. The result with different FISTA iterations can be seen in Table 3.2 (in the "rec coeff = 0" row). As predicted, there is a steady decrease in the reconstruction loss and an increase in sparsity. Clean accuracy doesn't change significantly, but robust accuracy increases. With 16 iterations, sparsity is above 90%, while also being more robust.

We saw in Section 3.2 that training with added reconstruction loss provides further improvement in robust accuracy. Thus I trained the same models with a reconstruction coefficient of 1 and 2 as well. The results can be seen in the same table. As expected, accuracy decreased once again, while robust accuracy increased. The best value for L_2 accuracy is 47.01%, which is in the edge case of 16 iterations and a reconstruction coefficient of 2. This is a 15.17% improvement over the baseline of 31.84%. Unfortunately, it is at the price of 3.4% decrease in clean accuracy over the baseline. Since this case also has a low reconstruction loss and a beneficially high sparsity of 96.06%, we should try to save this case by improving its clean accuracy. This can be done by a data augmentation technique called SaliencyMix, discussed in the next section.

Rec coeff	Iterations	Accuracy	Sparsity	Rec loss ($\downarrow$)	L_2 acc	L_∞ acc
0	2	93.96%	78.52%	0.7390	34.30%	0.07%
	4	93.77%	79.10%	0.4376	35.43%	0.07%
	8	93.87%	84.10%	0.3078	35.71%	0.20%
	16	93.54%	92.02%	0.2841	38.23%	0.49%
1	2	92.40%	64.14%	0.6637	39.92%	0.23%
	4	93.21%	66.17%	0.3789	37.88%	0.15%
	8	92.86%	93.77%	0.2085	42.41%	0.27%
	16	93.61%	91.68%	0.2107	41.33%	0.27%
2	2	92.24%	64.05%	0.6716	42.86%	0.41%
	4	92.82%	65.73%	0.3822	38.22%	0.41%
	8	92.65%	94.44%	0.2059	44.23%	0.74%
	16	91.93%	96.06%	0.2048	47.01%	0.55%

Table 3.2: Effect of different FISTA iterations and different reconstruction coefficients on SDNet-18 with $\lambda = 2$.

3.4 Improving clean accuracy with SaliencyMix

I chose a novel augmentation method, SaliencyMix (for details see Section 2.4) to improve the clean accuracy of the previous experiments. 50% of the batches are modified with SaliencyMix, similarly to the original SaliencyMix paper. With this modification the same experiments were run on SDNet-18 with $\lambda = 2$ as in Section 3.3. The results are in Table 3.3.

Rec coeff	Iterations	Accuracy	Sparsity	Rec loss ($\downarrow$)	L_2 acc	L_∞ acc
0	2	95.01%	79.10%	0.7366	34.70%	0.09%
	4	94.95%	79.05%	0.4252	27.43%	0.00%
	8	95.05%	83.94%	0.3122	30.30%	0.00%
	16	94.90%	91.97%	0.2889	37.09%	0.01%
1	2	93.54%	64.12%	0.6676	29.96%	0.10%
	4	94.47%	66.00%	0.3770	31.09%	0.00%
	8	94.53%	93.98%	0.2112	38.94%	0.04%
	16	94.47%	90.84%	0.2112	32.41%	0.02%
2	2	93.83%	64.28%	0.6562	38.54%	0.11%
	4	94.17%	65.87%	0.3778	35.11%	0.09%
	8	94.40%	94.30%	0.2060	41.38%	0.08%
	16	93.71%	94.91%	0.2020	36.45%	0.01%

Table 3.3: Effect of different FISTA iterations and different reconstruction coefficients on SDNet-18 with $\lambda = 2$ trained with SaliencyMix augmentation.

We can see an overall increase in clean accuracy by about 1-2%, which is enough to get us closer to the baseline results. The reconstruction loss and sparsity didn't change significantly, but unfortunately the robust accuracy decreased. It also seems that there is inconsistency in which iteration number provides the best robust accuracy, since in the previous example it was 16, but in this case it is 8 (in the reconstruction coefficient = 2 case). So for the SaliencyMix experiments, reconstruction coefficient of 2 and 8 iterations give our best result: 41.38% L_2 accuracy, which is a 9.54% improvement over the baseline of 31.84%, at the cost of only 0.93% decrease in clean accuracy. The sparsity improved the most, with a 44.30% increase, giving us a large sparsity of 94.30%.

3.5 Interpretability of the dictionary

In the SDNet paper, Dai et al. present a visualization of the hidden features of the network but neglect the learned dictionaries. In this section I show the dictionaries of the trained models, how they changed with different settings, and show an example of how they work. The first layer of the models have 3x3 kernels (for CIFAR-10), which makes visualization a harder task. For this reason, the following models were trained with a slightly different architecture - the first layer's kernels were changed to 7x7 (the architectures for Imagenet already have this kernel size). Other than this, no other changes were made.

On Figure 3.4 we can see 4 different trained model's first layer's dictionaries. They all have 64 kernels/atoms, as we know from the architecture diagram (Figure 2.1). The

first 2 models' dictionaries seem to have point-like atoms ((a) and (b)) and it is hard to find any structure in them. These models were trained without reconstruction loss. The difference is that (b) has a larger iteration number. As mentioned before, increasing the iteration number decreases the reconstruction loss - but it doesn't affect the dictionaries themselves. That is why there is no improvement in the dictionary of (b), despite being able to reconstruct the input better.

The next 2 dictionaries have both 16 iterations, but were also trained with added reconstruction loss. The (c) case with a coefficient of 1 and the (d) one with a coefficient of 2. Here we can see the other benefit of training with added reconstruction loss as hinted before. With (c) we can already see structures emerge and colors being separated. There are grayscale atoms that are horizontal, vertical, and different orientation edge detectors. Colors are separated from these, to pink, orange/blue and green atoms for example. These may be important since the dataset contains many outdoor images with sky and grass in them. All of these changes become more apparent in the case of (d), where even checkerboard structures emerge. Since this is the first layer of the network, we shouldn't expect anything more complicated than edges and colors templates. This is why an important continuation of this work will be the extension to SDNet-18-All, where deeper layer will also be sparse, and may provide dictionaries that are more interesting and humanly interpretable structures.

An example of dictionary (d) reconstructing an image can be seen in Figure 3.5. The input image (a) is a ship. If we apply a convolution with our dictionary on it, we will get the activations in (c). White represents zero values, and we can see how sparse the representation really is. Since the proximal operator is non-negative, there are only positive values that are shown with blue. In (d) we can see what each dictionary reconstructs. For example, the input ship has a red part on its body. This is reconstructed by the atom in column 6 and row 4. If we check the same column and row on grid (d), we can see a red line - this is reconstructed by that one atom. Similarly, the blue sky and sea are made by mixing different shades of blue. Since color and structures are mostly separated, the details are added by the edge detector atoms. Finally, if we add up all of these small reconstructions, we will get the reconstructed image (e).

We can see why sparse coding gives us interpretability - the sparse model tells us that we can reconstruct any image with a global dictionary. If the dictionary atoms have structure, the model will be able to deconstruct and reconstruct any image giving us deep insights about it and the model as well.

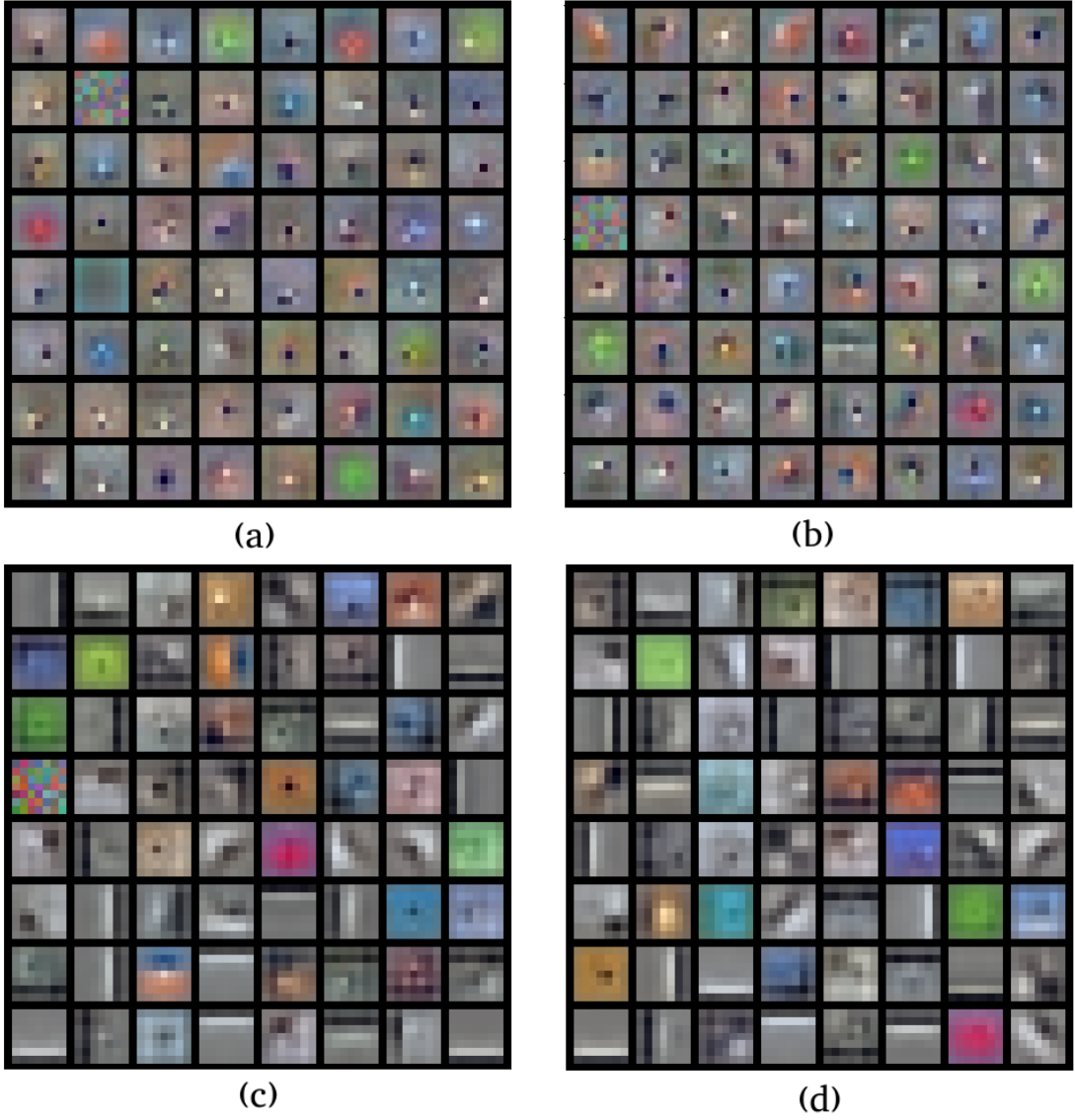


Figure 3.4: Dictionaries of the 4 trained SDNet-18 models: (a) No reconstruction loss, 2 iterations, (b) No reconstruction loss, 16 iterations, (c) Reconstruction loss with coefficient 1, 16 iterations, and (d) Reconstruction loss with coefficient 2, 16 iterations.

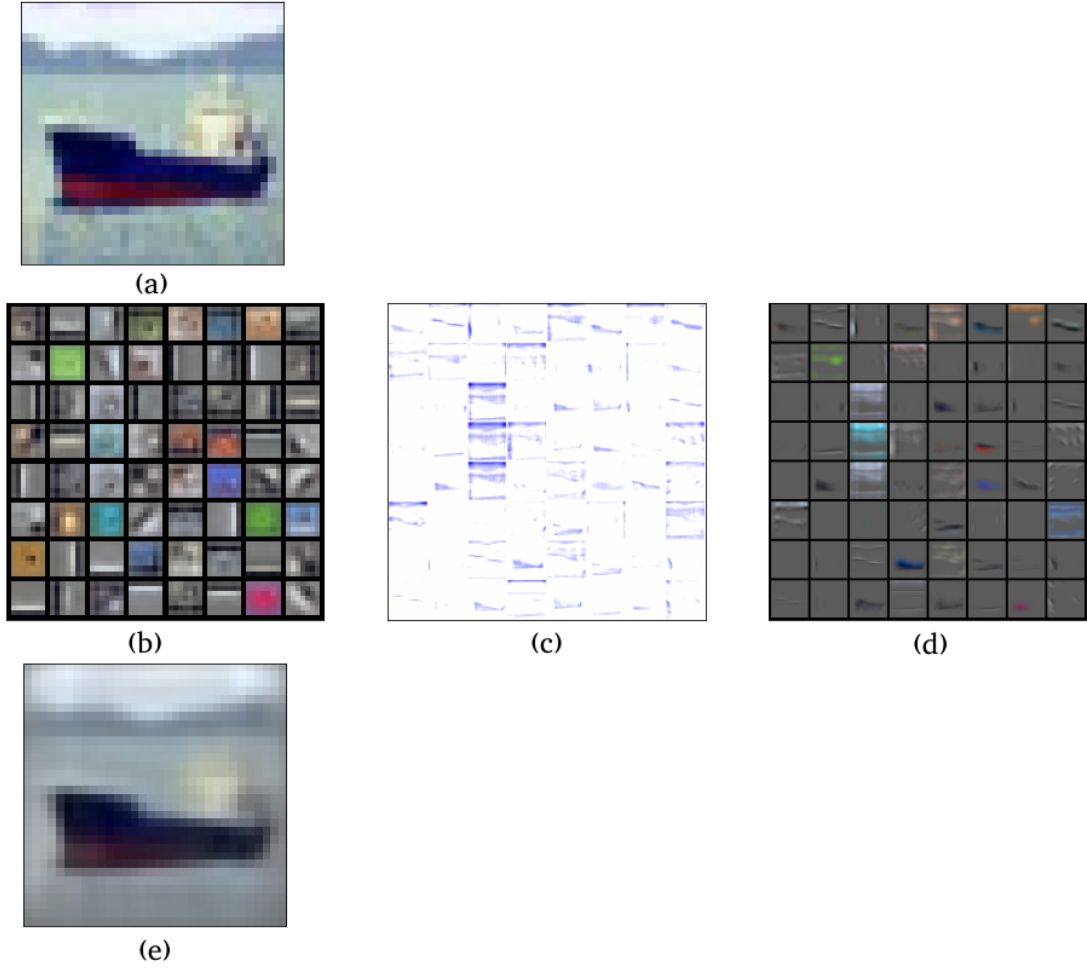


Figure 3.5: Visualization of how the dictionary works, with the dictionary from Figure 3.4(d): (a) Input image, (b) dictionary, (c) activations of each dictionary (kernel) - white is zero, while blue are positive values -, (d) reconstructed input if only the atom in the same position is used, and (e) reconstructed input.

Chapter 4

Conclusion

In this work I have analysed and made several improvements to the SDNet proposed by Dai et al. Most importantly, I have improved the SDNet-18's sparsity greatly from 50% to 94.3% which is crucial for theoretical stability guarantees and improved its adversarial L_2 robustness from 31.84% to 41.38% which is highly relevant for the practical usage of the network. Although these would otherwise be at the cost of classification performance (for example in the case of adversarial training), I used a novel augmentation method, SaliencyMix, to maintain a similar 94.4% accuracy. Without SaliencyMix - although at the cost of a lower accuracy of 91.93% - I have shown that a 47.01% robust L_2 accuracy can be also achieved. Finally, I have analyzed SDNet18's dictionary and have shown that it can be made more structured using a modified loss function, and have also shown a way to interpret the workings of the CSC layer.

There are several ways in which I wish to continue and expand this work. As the natural continuation of this work I wish to extend it to the SDNet-18-All. For SDNet-18-All several difficulties emerge, like finding the optimal λ value, since the sparsity of every layers depends on other layers' λ values. K-sparsity [33] could solve this issue, by simplifying hyper-parameter tuning to finding only one k value for each layer. In fact, we can make this k value depend on the sparsity that we wish to achieve, which would make it easier to find the right level of sparsity. Another difficulty with SDNet-18-All is that the reconstruction loss for it is ambiguous: it could be defined just for the CSC layers, just for blocks, or the whole network as well. All of these approaches need to be considered in order to achieve similar results as in this work.

Another continuation of this work is the extension to the Imagenet dataset [35]. In this case, more complex architectures would be needed to achieve high accuracy.

A final idea for the expansion of this work is generalizing the SDNet to other architectures. To create SDNet, Dai et al. used an existing baseline neural network and they replaced its convolutional layers with CSC layers. State-of-the-art models such as the Vision Transformer [36] could be used as a base model and sparsified. The Vision Transformer already works with image patches, which the CSC layers could perhaps reconstruct easier. Even other domains could benefit from this idea, since - as mentioned before - adversarial attacks can be applied to many types of data and models.

Acknowledgements

I would like to thank Aslan Mahmoud and my supervisor, András Lőrincz for their help, support and constructive criticism over the past year. Without their guidance and professional competence these results would not have been possible.

Bibliography

- [1] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [2] Christian Szegedy et al. “Inception-v4, inception-resnet and the impact of residual connections on learning”. In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.
- [3] Tsung-Yi Lin et al. “Feature pyramid networks for object detection”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 2117–2125.
- [4] Jacob Devlin et al. “Bert: Pre-training of deep bidirectional transformers for language understanding”. In: *arXiv preprint arXiv:1810.04805* (2018).
- [5] Christopher D Manning et al. “The Stanford CoreNLP natural language processing toolkit”. In: *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*. 2014, pp. 55–60.
- [6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. “Explaining and harnessing adversarial examples”. In: *arXiv preprint arXiv:1412.6572* (2014).
- [7] Kaidi Xu et al. “Interpreting adversarial examples by activation promotion and suppression”. In: *arXiv preprint arXiv:1904.02057* (2019).
- [8] Kaidi Xu et al. “Structured adversarial attack: Towards general implementation and better interpretability”. In: *arXiv preprint arXiv:1808.01664* (2018).
- [9] Han Wu et al. “Adversarial Driving: Attacking End-to-End Autonomous Driving”. In: *arXiv preprint arXiv:2103.09151* (2021).
- [10] Hongge Chen et al. “Attacking visual language grounding with adversarial examples: A case study on neural image captioning”. In: *arXiv preprint arXiv:1712.02051* (2017).

- [11] Pin-Yu Chen et al. “Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models”. In: *Proceedings of the 10th ACM workshop on artificial intelligence and security*. 2017, pp. 15–26.
- [12] Alexey Kurakin, Ian J Goodfellow, and Samy Bengio. “Adversarial examples in the physical world”. In: *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018, pp. 99–112.
- [13] Aleksander Madry et al. “Towards deep learning models resistant to adversarial attacks”. In: *arXiv preprint arXiv:1706.06083* (2017).
- [14] John X Morris et al. “Textattack: A framework for adversarial attacks, data augmentation, and adversarial training in nlp”. In: *arXiv preprint arXiv:2005.05909* (2020).
- [15] Chiyuan Zhang et al. “Understanding deep learning (still) requires rethinking generalization”. In: *Communications of the ACM* 64.3 (2021), pp. 107–115.
- [16] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *arXiv preprint arXiv:1312.6199* (2013).
- [17] Florian Tramèr et al. “Ensemble adversarial training: Attacks and defenses”. In: *arXiv preprint arXiv:1705.07204* (2017).
- [18] Nicolas Papernot et al. “Distillation as a defense to adversarial perturbations against deep neural networks”. In: *2016 IEEE symposium on security and privacy (SP)*. IEEE. 2016, pp. 582–597.
- [19] Nicolas Papernot et al. “Practical black-box attacks against machine learning”. In: *Proceedings of the 2017 ACM on Asia conference on computer and communications security*. 2017, pp. 506–519.
- [20] Kwan Ho Ryan Chan et al. “ReduNet: A white-box deep network from the principle of maximizing rate reduction”. In: *The Journal of Machine Learning Research* 23.1 (2022), pp. 4907–5009.
- [21] Dávid Szeghy et al. “Structural Extensions of Basis Pursuit: Guarantees on Adversarial Robustness”. In: *arXiv preprint arXiv:2205.08955* (2022).
- [22] Yaniv Romano et al. “Adversarial noise attacks of deep learning architectures: Stability analysis via sparse-modeled signals”. In: *Journal of Mathematical Imaging and Vision* 62 (2020), pp. 313–327.

- [23] Mingyang Li et al. “Revisiting sparse convolutional model for visual recognition”. In: *Advances in Neural Information Processing Systems* 35 (2022), pp. 10492–10504.
- [24] Michael Elad. *Sparse and redundant representations: from theory to applications in signal and image processing*. Vol. 2. 1. Springer, 2010.
- [25] David L Donoho and Michael Elad. “Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization”. In: *Proceedings of the National Academy of Sciences* 100.5 (2003), pp. 2197–2202.
- [26] Amir Beck and Marc Teboulle. “A fast iterative shrinkage-thresholding algorithm for linear inverse problems”. In: *SIAM journal on imaging sciences* 2.1 (2009), pp. 183–202.
- [27] Vardan Pappyan, Yaniv Romano, and Michael Elad. “Convolutional neural networks analyzed via convolutional sparse coding”. In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 2887–2938.
- [28] Kaiming He and Jian Sun. “Convolutional neural networks at constrained time cost”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 5353–5360.
- [29] Junsuk Choe and Hyunjung Shim. “Attention-based dropout layer for weakly supervised object localization”. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 2219–2228.
- [30] Sangdoo Yun et al. “Cutmix: Regularization strategy to train strong classifiers with localizable features”. In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6023–6032.
- [31] AFM Uddin et al. “Saliency mix: A saliency guided data augmentation strategy for better regularization”. In: *arXiv preprint arXiv:2006.01791* (2020).
- [32] Matthew D Zeiler et al. “Deconvolutional networks”. In: *2010 IEEE Computer Society Conference on computer vision and pattern recognition*. IEEE. 2010, pp. 2528–2535.
- [33] Alireza Makhzani and Brendan Frey. “K-sparse autoencoders”. In: *arXiv preprint arXiv:1312.5663* (2013).

- [34] Vardan Papyan, Jeremias Sulam, and Michael Elad. “Working locally thinking globally: Theoretical guarantees for convolutional sparse coding”. In: *IEEE Transactions on Signal Processing* 65.21 (2017), pp. 5687–5701.
- [35] Jia Deng et al. “Imagenet: A large-scale hierarchical image database”. In: *2009 IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.
- [36] Alexey Dosovitskiy et al. “An image is worth 16x16 words: Transformers for image recognition at scale”. In: *arXiv preprint arXiv:2010.11929* (2020).

List of Figures

2.1	The ResNet-18, SDNet-18 and SDNet-18-All architectures	8
2.2	Examples of different reconstruction loss values	11
3.1	Trained SDNet-18's robust accuracy with different biases	14
3.2	The SDNet models' sparsities	15
3.3	The SDNet-18 model's accuracies, sparsity and reconstruction loss when trained with different λ values	16
3.4	Dictionaries of the 4 trained SDNet-18 models	21
3.5	Visualization of how the dictionary works	22

List of Tables

3.1	Accuracy results for the baseline models.	14
3.2	Effect of different FISTA iterations and different reconstruction coefficients on SDNet-18 with $\lambda = 2$	18
3.3	Effect of different FISTA iterations and different reconstruction coefficients on SDNet-18 with $\lambda = 2$ trained with SaliencyMix augmentation. .	19