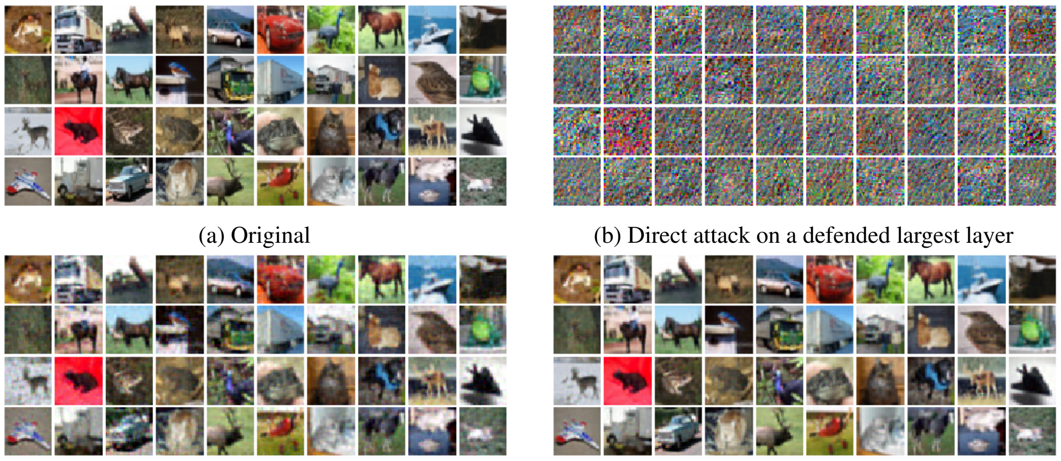
Conv2d(in_channels=3, out_channels=32, kernel_size=3, stride=1, padding=1)
ReLU()
AvgPool2d(kernel_size=2, stride=2)
Conv2d(in_channels=32, out_channels=64, kernel_size=1, padding=1)
nn.ReLU()
AvgPool2d(kernel_size=2, stride=2)
Linear(in_features=5184, out_features=2000)
ReLU()
Linear(in_features=2000, out_features=1000)
ReLU()
Linear(in_features=1000, out_features=10)

Table 2: ConvBig architecture.



(a) Original

(b) Direct attack on a defended largest layer

(c) Our attack on the defended largest layer

(d) Our attack on a defended second largest layer

Figure 5: We compare the results of a direct attack on the defended network 5b with our attack 5c. Furthermore we show our attack on a network in which the second largest layer is defended 5d. All networks have been trained for 10 steps.

A Appendix

Here we provide additional details for our work.

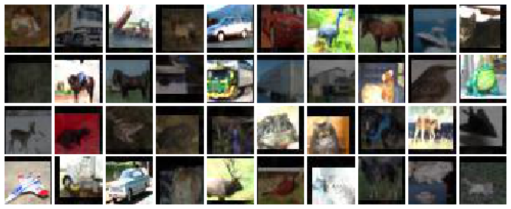
A.1 Soteria

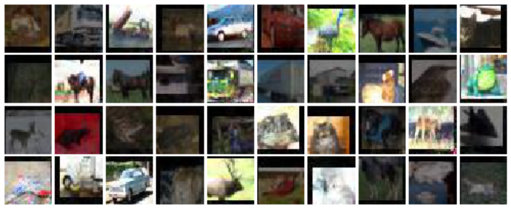
For Soteria, we built directly on the Sun et al. [17] repository using their implementation of the defense and the included Inverting Gradient [4] library. Testing was conducted primarily on the ConvBig architecture presented below:

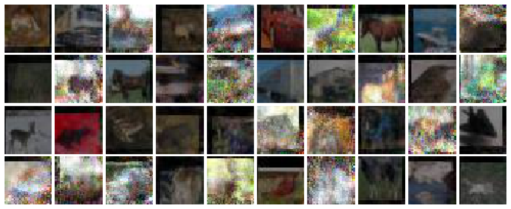
The architecture consists of a convolutional "feature extractor" followed by three linear layers. We chose this architecture because (i) it is simple in structure while providing reasonable accuracies on datasets such as CIFAR10 and (ii) because (unlike many small convolutional networks) it has more than one layer with a significant fraction of the overall network parameters. In particular, the first linear layer roughly contains around 80% of the network weights and the second one 20%. This is particularly relevant for our attack, as cutting out a large layer of weights will negatively affect the reconstruction quality. As it is in practice uncommon to have the majority of weights in a single layer, we believe our architecture provides a reasonable abstraction. To justify this, we show in Fig. 5 that no matter which of the larger two layers is defended by Soteria, we can attack the network successfully. For all Soteria defenses we set the pruning rate (refer to [17] for details) to 80%. Note however that our attack works independent of the pruning rate, as we always remove the entire layer.

Conv2d(3, 1 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
Conv2d(1 * width, 2 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
Conv2d(2 * width, 2 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
Conv2d(2 * width, 4 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
Conv2d(4 * width, 4 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
Conv2d(4 * width, 4 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
MaxPool2d(3),
Conv2d(4 * width, 4 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
Conv2d(4 * width, 4 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
Conv2d(4 * width, 4 * width, kernel_size=3, padding=1), BatchNorm2d(), ReLU()
MaxPool2d(3)
Linear(36 * width, 3)

Table 3: ConvNet architecture. Our benchmark instatiation uses a width of 64.


(a) Reconstruction after 5 training step


(b) Reconstruction after 10 training steps


(c) Reconstruction after 20 training steps

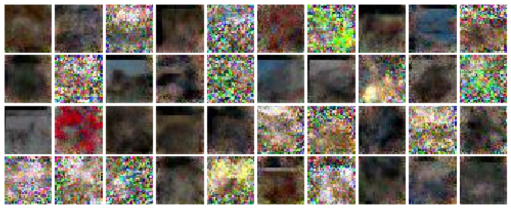

(d) Reconstruction after 50 training steps

Figure 6: Our reconstruction results after several training steps with ATS, batch_size 32, and augmentations 7-4-15. We can see how the visual quality starts to decline after 10 steps and at 50 steps one can no longer reliably recover the input.

We use the Adam optimizer with a learning rate of 0.1 with decay. As similarity measure, we use cosine similarity. Besides cutting one layer of weights, we weigh all gradients equally. The total variation regularization constant is 4×10^{-4}. We initialize the initial guess randomly and only try reconstruction once per image, attacking one image at a time. For training, we used a batch size of 32. We trained the network without defense and applied the defense at inference time to speed up the training.

Appendix B: Automated Transformation Search For ATS, we built upon the repository released alongside [3]. We use the ConvNet architecture with a width of 64 also proposed in [3] and train with the augmentations "7-4-15", "21-13-3", "21-13-3+7-4-15" which perform the best on ConvNet with CIFAR100. We present the ConvNet architecture in Table 3.

For reconstruction, we use the Adam optimizer with a learning rate of 0.1 with decay. As similarity measure, we use cosine similarity weighing all gradients equally. The total variation regularization constant is 1×10^{-5}. We initialize the initial guess randomly and only try reconstruction once per image, attacking one image at a time. For training, we used a batch size of 32 and trained individually for every set of augmentations.

In Figure 6 we show how the quality of the reconstructed inputs degrades during training. Nevertheless we can recover high-quality inputs during the first 10 to 20 training steps.

A.2 Parameters for the attacks

All attacks are implemented using the code from Geiping et al. [4], and using cosine similarity between gradients, the Adam optimizer [6] with a learning rate of 0.1 for both, a total variation regularization of 10^{-5} for ATS and 4×10^{-4} for Soteria, as well as 2000 and 4000 attack iterations respectively. We perform the attack on both networks using batch size 1.

Since the range of β for which the different attacks perform well is wide, prior to the grid search we need to calculate a range of reasonable values for β for each of the attacks. We do this by searching for values of β for which the respective attack is optimal. The rest of the parameters of these attacks are set to the same set of initial values. The values of β considered are in the range $[1 \times 10^{-7}, 1 \times 10^{5}]$ and are tested on logarithmic scale. The final range for β used in the grid search is given by the range $[0.5\beta_*, 2\beta_*]$, where β_* is the values that produced the highest PSNR.