Bachelor thesis
Computing Science

Radboud University

Exploring Unlearnable Examples

Author:
Tijn Berns
s1027659

First supervisor:
MSc. Zhuoran Liu
Z.Liu@cs.ru.nl

Second supervisor:
MSc. Alex Kolmus
A.Kolmus@cs.ru.nl

First assessor:
Prof. Martha Larson
M.Larson@cs.ru.nl

Second assessor:
Prof. Tom Heskes
Tom.Heskes@ru.nl

June 3, 2021

Abstract

The large amount of publicly accessible image data has been key to the success of machine learning. However, the publicly accessible data also raises privacy concerns about unauthorized data exploitation. To protect personal images from unauthorized neural network training, error-minimizing noise has been proposed. The error-minimizing noise can be added to personal images making them unusable for training neural networks. Therefore, these images are also referred to as unlearnable examples. This thesis aims to improve the performance of deep neural networks trained on data with added error-minimizing noise, and thereby give a direction on how to improve the noise. The research explores the strength of error-minimizing noise by focusing on the following two aspects: its resistance against adversarial training, and its resistance against data augmentation. We show that the effects of the noise are highly dependent on the presence of color by conducting experiments on grayscale images. We demonstrate that generating error-minimizing noise on grayscale image does not improve its resistance against grayscale transformations. Furthermore, we verify that the effect of error-minimizing noise can be compensated by using adversarial training.

Contents

Chapter 1

Introduction

The rise in computing resources and the growing amount of publicly accessible image data are the foundations of the progress that has been made in research related to neural networks. However, the large amount of publicly accessible image data also raises concerns regarding how the images are collected. A party could easily create a dataset by collecting images from social media platforms, without asking for the consent of the owners of the images. A real-world examples of this is the MegaFace dataset. This is a facial dataset consisting of millions of images taken from Flickr without the owners being aware of it. This example shows that the need for data is colliding with the privacy of internet users.

Luckily there is awareness for the privacy and data protection concerns in research related to machine learning [11][16][19][22][23]. Huang et al.[7] made a great step to protect image data from unauthorized machine learning by proposing a method to generate error-minimizing noise. The goal of this noise is to make images someone uploads to the internet deteriorate the training process of a neural network. Therefore, images with such noise added to them are referred to as unlearnable examples. Since these unlearnable examples deteriorate the training process of neural networks, they are less useful and therefore the chance of the images being collected without consent is reduced. Although the method is a good direction in achieving its goal, it has not been shown that the error-minimizing perturbations truly make images unusable for training neural networks. In this research we explore the unlearnable examples by experimenting with their resistance against adversarial training and data augmentation.

Huang et al. show that the noise is fairly resistant against adversarial training under specific conditions. However, in these conditions, the scale of the error-minimizing perturbations is much larger than that of the adversarial noise added during training. This motivates us to further explore the resistance of error-minimizing perturbations against adversarial training.

It has also been shown that the error-minimizing perturbations are re-

sistant against four data augmentation techniques: Cutout [2], Mixup [27], Cutmix [26], and Fast Augmentation [15]. The resistance against other data augmentation techniques like JPEG compression, which is known to offer protection against small adversarial perturbations [3], has not been tested. Therefore, we are motivated to further explore the resistance of the error-minimizing noise against data augmentation.

Before we can apply the error minimizing noise in real real-world scenarios, we must ensure that the method makes data unusable under a wide variety of training methods. The goal of the research is to show the weak points of the error-minimizing noise and thereby give a direction on how to improve the method. To show the weak points, we have to improve the performance of neural networks trained on data with added error-minimizing noise. This translates to the following research question:

How can we improve the performance of deep neural networks trained on data with added error-minimizing noise?

To answer this research question we focus on adversarial training and data augmentation. To test the effects of adversarial training, we train using an FGSM-, or PGD-based objective function on an error-minimizing perturbed dataset. We test the effects of data augmentation by conducting experiments with the following data augmentations: JPEG compression, grayscale transformation, course dropout, median blur, and Gaussian noise. The research makes the following contributions:

- We show that error-minimizing noise as proposed by Huang et al. [7] is highly dependent on color. By transforming images with added error-minimizing noise into grayscale images, the effectiveness of the noise significantly drops. The effectiveness of the noise can be further degraded by combining grayscale transformation with applying JPEG compression or adding Gaussian noise.

- We show that generating error-minimizing noise on grayscale images does not make the noise much more resistant against grayscale transformations during training. Similar to the original error-minimizing noise, the combination of grayscale transformation with JPEG compression or Gaussian noise effectively counteracts the effects of error-minimizing noise generated on grayscale images.

- We verify that adversarial training improves the performance of models trained on data with added error-minimizing noise. Using PGD-style adversarial training on error-minimizing perturbed datasets, a model can achieve similar performance to models adversarial trained on clean datasets.

Chapter 2

Related Work & Preliminaries

In this chapter, we cover existing literature and discuss principles that we will use in the remainder of the thesis. First, two basic mathematical principles are explained. Second, relevant work and principles of neural networks are explained. After this, we introduce the concept of adversarial attacks and training. Finally, we explain how error-minimizing noise is generated, and describe the knowledge gap.

2.1 Mathematical Foundations

Gradients

A gradient is vector containing the partial derivates of a multi-dimensional function with respect to all of its variables. Like derivates of a one-dimensional function, the gradient gives us the slope of a function at a given point. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a n-dimensional function, and $\boldsymbol{x} = [x_0, x_1, \ldots, x_{n-1}]$ be the input to function f. The gradient of f with respect to input $\boldsymbol{x}$, denoted by $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$, is computed as follows:

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \left[\frac{\partial f(\boldsymbol{x})}{\partial x_0}, \frac{\partial f(\boldsymbol{x})}{\partial x_1}, \ldots, \frac{\partial f(\boldsymbol{x})}{\partial x_{n-1}} \right]^\top . \tag{2.1}$$

Gradients are useful for the optimization of neural networks. They are used to determine how to update the parameters of a network. The exact process for this optimization as well as a definition of network parameters is given in Section 2.2.2.

Norms

A norm is a kind of measure for the size of a mathematical object. Throughout this thesis we use the L^p-norm, which is a popular norm used in the field

of adversarial machine learning [5]. In this thesis, the norm is used to compute the difference between a clean image and an image containing noise. Let $\boldsymbol{x}$ be an vector of n elements representing an image, then the L^p-norm for $\boldsymbol{x}$ is defined as follows:

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=0}^{n-1} |x_i|^p \right)^{\frac{1}{p}}. \tag{2.2}$$

Norms that are often used are the L^1-, L^2-, and the L^∞-norm. In our experiments we use the L^∞-norm, which is equivalent to taking the maximum element of the vector. The L^∞-norm one of the optimal norms for this task [5] and is therefore commonly used when generating adversarial perturbations.

2.2 Neural Networks

Neural networks are commonly used for classification, regression, and clustering. In this research, we will focus on neural networks used as a classifier. A neural network consists of a sequence of layers, where each layer consists of neurons. The general structure of a neural network can be seen in Figure 2.1.

A neural network consists of one input layer, $k \geq 0$ hidden layers, and one output layer. Neurons in one layer are mapped to neurons in the proceeding layer via weight values. Each neuron in the network computes the weighted sum of all of its inputs. If the resulting value is below a certain threshold it does not pass its value to the next layer. This threshold is determined by an activation function. A bias is added to each neuron, shifting the activation of that neuron. For example, when a positive bias is added to a neuron, the threshold shifts down, making the neuron activate at lower values. The weighted sum with added bias is what is given as input to the activation function. We refer to all the weights and biases in the network as the network parameters.

In case the model is used for classification, the values of neurons in the output layer correspond to the probability the input data belongs to a specific class. Take for example an output node i with a value of 0.7, this implies there is a chance of 0.7 the input corresponds to class i.

2.2.1 Classification

A model classifies input by forwarding it through its layers. Let us demonstrate the calculations done during the classification of input x using 2 layers of a neural network, $a^{(0)}$ and $a^{(1)}$. For demonstration purposes, assume both layers consist of m neurons. The computation of the value of neuron i in

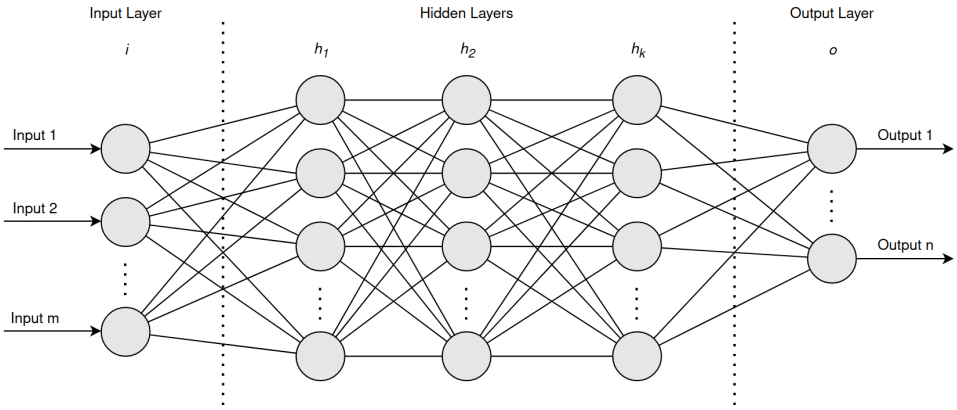


Figure 2.1: General structure of a neural network

layer $a^{(1)}$ is as follows:

$$a_i^{(1)} = \sigma(a_0^{(0)} \cdot w_{(0,i)} + a_1^{(0)} \cdot w_{(1,i)} \cdot \ldots \cdot a_m^{(0)} w_{(m,i)} + b_i). \tag{2.3}$$

In this formula, σ denotes a squishing function, $\sigma : \mathbb{R} \to (0, 1)$. The weight value between neurons $i \in a^{(0)}$ and $j \in a^{(1)}$ is denoted by $w_{i,j}$. The bias of neuron $a_i^{(1)}$ is denoted by b_i. The same computation is repeated for each pair of adjacent layers from the input layer to the output layer. If we rewrite the weights between all neurons of two layers as a matrix $\mathbf{W}$, the inputs and biases as vectors $\mathbf{a}$ and $\mathbf{b}$ respectively, we can write the full transition from a layer to the next layer in the following, more compact form:

$$\mathbf{a}^{(n+1)} = \sigma(\mathbf{W}\mathbf{a}^{(n)} + \mathbf{b}). \tag{2.4}$$

2.2.2 Training

The parameters of a network are optimized during the training of the network, commonly stochastic gradient descent is used for this. Before we give a formal description of this process, we introduce some notations. We will use these notations throughout the remainder of the thesis. First, let us define $\mathcal{X}$ as the input space of a model. We define a model, $h_\theta : \mathcal{X} \to \mathbb{R}^k$, as a mapping from input space to a k-dimensional output vector, where k is the number of classes, and θ the parameters of the model.

Second, we define the loss function, $J : \mathbb{R}^k \times \mathbb{Z}_+ \to \mathbb{R}_+$, as a mapping from model predictions and true labels to a positive real number. The loss of model h_θ on input x, with true labels y, can now be defined as $J(h_\theta(x), y)$. This tells us how far the model's predictions are from the true class labels. A high loss indicates the predictions of the model are far from the true class labels. Commonly the cross-entropy loss is used as the loss function.

Now that we have defined the model and loss function we are able to give a more formal description of how stochastic gradient descent updates the model parameters. It does so by minimizing the loss function, performing the following update each time a batch is forwarded through the model:

$$\theta_{i+1} = \theta_i - \alpha \nabla_\theta J(h_\theta(x_r), y_r), \tag{2.5}$$

where $h_\theta(x_r)$ denotes the model prediction of a random sample from the batch, y_r the true class label corresponding to sample x_r, and α the step size, also referred to as the learning rate. The learning rate determines how much we want the model parameters to change. To compute the gradient of the loss function, we use the chain rule from calculus to compute the derivative of the model parameters. This is done in reverse order, from the output layer to the input layer. The loss is propagated back through the model, which is why the method of parameters optimization is referred to as backpropagation [13].

When we are close to a local minimum, a small learning rate is preferred to prevent stepping over the local minimum. This is why we want to lower the learning rate as the model converges. There are numerous scheduling methods available that update the learning rate of stochastic gradient descent during the training of a network. In our experiments we use a cosine annealing scheduler [17]. The strategy this scheduler uses is to let the learning rate drop rapidly, after which it is increased rapidly. This enlarging of the learning rate simulates a restart of the learning process with better model parameters. The resetting of the learning rate is also referred to as a warm restart.

2.2.3 Convolution and Pooling

A convolutional neural network, or CNN, uses the fact that pixels that are close to each other within a picture, are related to one another. The first published CNN which got attention was the model introduced by LeCun et al. [14]. The model was designed specifically for learning from image data.

The main idea of a convolutional layer is to slide a $m \times n$ filter, also called a kernel, over the input tensor. We refer to a convolutional layer with a kernel of size $m \times n$, as an $m \times n$ convolutional layer. The values of the output tensor are computed by taking the dot product between the values in the kernel, and the values of the input image the kernel is currently considering. In Figure 2.2 an example of a computation of a 2×2 convolutional layer on a 3×3 image is shown. In this example, the top-left value of the output image is computed as follows:

$$0 \cdot 1 + 1 \cdot 3 + 2 \cdot 4 + 5 \cdot 6 = 0 + 3 + 8 + 18 = 29$$

It is possible to have multiple kernels in a single convolutional layer. If this is the case we do the same computation as in Figure 2.2 for each of the

kernels. Each computation results in a matrix. These matrices are summed and returned as the result of the convolutional layer with multiple kernels.

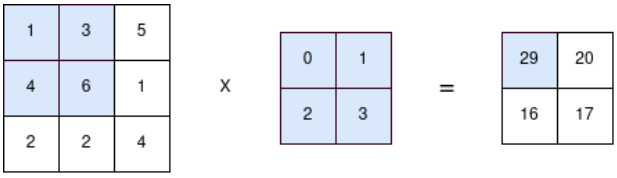


Figure 2.2: Example of a 2×2 convolutional layer. The input image is of size $1 \times 3 \times 3$.

Pooling layers are specific cases of convolutional layers and are used to reduce the size of the representation. A pooling layer achieves this by sliding a kernel over the input image, performing an aggregation on the pixels under the kernel. Two commonly used aggregations are taking the average, and taking the maximum of all pixels under the kernel. The pooling layers that use these aggregations are called average-pooling layers and max-pooling layers respectively. There are more aggregations we can perform, but since we will not be using them in our experiments, we do not list them here.

2.2.4 Batch Normalization

The concept of a batch-normalization layer is proposed by Sergey Ioffe and Christian Szegedy [9]. It is designed to accelerate and stabilize the training of neural networks. The motivation behind the design of batch normalization is the claim that neural networks suffer from internal covariate shifts. This is the same as saying a neural network suffers from changes in input distribution of the hidden layers.

When training a model the distribution of the inputs of hidden layers can change due to changes made in the model parameters. Batch normalization layers counteract this effect, by normalizing all of its input data to a unit normal distribution, i.e. a distribution with zero mean and a standard deviation of one. The batch normalization layer does this by estimating the mean and standard deviation of the batch that is forwarded through the model, these are the parameters of a batch normalization layer that are learned. The data is normalized by subtracting the mean from the data and dividing the data by the standard deviation. Note that this is the most basic implementation of a batch normalization layer.

2.2.5 ResNet18

In our experiments, we will be using a specific neural network referred to as ResNet18. The model was found by He et al. [6] and uses the same structure of residual blocks as the model which won the ImageNet Large Scale Visual

Recognition Challenge in 2015 [21]. The ResNet18 model consists of a 7×7 convolutional layer, followed by a batch normalization layer and eight residual blocks. The last residual block is followed by a fully connected layer of size 512. A fully connected layer is a standard layer as in Figure 2.1, in which each neuron is connected to each neuron in the previous layers.

A residual block as considered by He et al. [6], consists of two 3×3 convolutional layers followed by a batch normalization layer, and a ReLU activation function. The input of the residual block is added directly before the final ReLU function. The ReLu function is defined as follows:

$$f(x) = \max(0, x), \tag{2.6}$$

Both convolutional layers in a residual block have the same number of output channels. To change the number of channels, before the addition, a 1×1 convolutional layer is introduced. This convolution before the addition is done in the third, fifth, and seventh residual blocks.

In total, the model consists of 8 residual blocks, a 7×7 convolutional layer, and a fully connected layer. Each residual block consists of two convolutional layers. This adds up to a total of 18 layers, which is why the ResNet18 model is named as such.

2.3 Adversarial Attacks

In 2014, Szegedy et al. [24] found that deep neural networks can be confused by adding adversarial generated noise to the inputs of the network. We describe two adversarial attack methods that can be used to generate noise, which if added to the input data, maximize the loss of a neural network on the perturbed data.

Both methods are based on the claim that many small changes in input can add up to a large change in output. Say we have an adversarial example $\tilde{x} = x + \delta$ corresponding to a natural example x. When forwarding $\tilde{x}$ through a linear model, the dot product between a weight vector $\mathbf{w}$ and the adversarial example $\tilde{x}$ is computed. This is computed as follows:

$$\mathbf{w}^\top \tilde{x} = \mathbf{w}^\top x + \mathbf{w}^\top \delta. \tag{2.7}$$

From Equation 2.7 follows that perturbation δ causes the activation to grow by $\mathbf{w}^\top \delta$. If we use the L^∞-norm as a distance metric, and our perturbation is bounded by $\|\delta\|_\infty \leq \epsilon$, for some $\epsilon \in \mathbb{R}$, the activation can be maximized by assigning $\delta = \epsilon \operatorname{sign} \mathbf{w}$. If $\mathbf{w}$ has n dimensions and the average of the absolute values of the weight vector is m, the activation will grow by $\epsilon \cdot m \cdot n$.

Since $\|\delta\|_\infty$ does not grow with the dimensionality, but the activation does, a simple linear model can have adversarial examples if its input has sufficient dimensionality. This and the claim of Goodfellow et al.[5] that neural networks are too linear to resist linear adversarial perturbations, are the cornerstone for both FGSM and PGD.

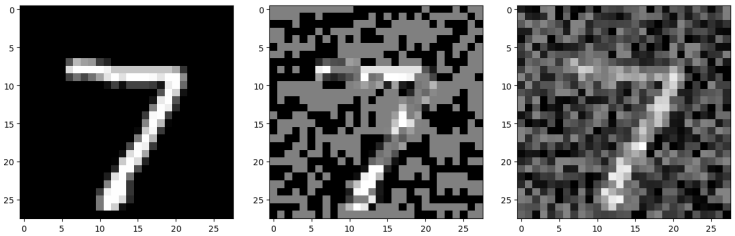
2.3.1 Fast Gradient Sign Method

The fast gradient sign method, or FGSM, was first described by Goodfellow et al. [5]. As the name suggests, the method is based on the gradient of the loss function. To maximize the loss, we change the parameters of the model by setting a single step of predefined size ϵ in the direction of the gradient of the loss function with respect to the input. Note how this is opposite to Equation 2.5, where we try to minimize the loss by subtracting the gradient of the loss function with respect to the model parameters.

Let us describe the process of generating the adversarial noise FGSM in a more formal way. We define x as the input to the model, and y the labels corresponding to input x. The goal of the adversarial attack is to find δ, such that δ maximizes $J(h_\theta(x+\delta), y)$. Let ϵ be the allowed perturbation, then the fast gradient sign method computes δ as follows:

$$\delta = \epsilon \operatorname{sign}(\nabla_x J(h_\theta(x), y)). \tag{2.8}$$

Because we are using the sign function, each input has either ϵ added to or subtracted from its original value. The effect can be seen in Figure 2.3b, especially in the dark area of the image, where we can only see two different grayscale values. By using the sign function we can increase the loss by a significant amount while only having to step once in the direction of the gradient. This means that to compute δ for an input x, we only need to backpropagate through the model once, which makes the method computationally efficient.



(a) Natural (b) FGSM (c) PGD

Figure 2.3: Example from the MNIST dataset: (a) shown without perturbation, (b) with perturbation generated by FGSM, (c) with perturbation generated by PGD. For both, PGD and FGSM, $\epsilon = 0.3$.

2.3.2 Projected Gradient Descent

Projected Gradient Descent, or PGD, is another method that can be used to generate adversarial noise. It is more powerful than the previously described fast gradient descent method, as concluded by Madry et al. [18]. PGD is

essentially a multi-step variant of FGSM which computes δ at iteration $t+1$ in the following way:

$$\delta^{t+1} = \mathcal{P}(\delta^t + \alpha \operatorname{sign}(\nabla_x J(h_\theta(x + \delta^t), y))), \tag{2.9}$$

where $\mathcal{P}$ denotes the projection onto the desired ball of interest, e.g. clipping in the case of L^∞, and α denotes the step size. The step size tells us how much we want delta to change each iteration. Choosing the step size is extremely important, as choosing a too large step size results in the method taking a step too large in the direction of the boundary, making the method act like FGSM. Choosing a too small step size results in the method exploring only a small section of the loss landscape, reducing the chance of finding good local maxima. To prevent these problems the step size is often a small fraction of ϵ, e.g. $\alpha = \epsilon/10$.

To improve the performance of PGD, we can run PGD multiple times from random initial noise. By doing so, PGD can find different local maxima in the loss landscape. We set δ to the perturbation resulting in the highest local maxima. In Figure 2.3c, an MNIST example with adversarial noise generated using PGD with four random restarts each with ten iterations is shown. We can see that the noise consists of more than two different grayscale values and therefore looks more random than the noise generated by FGSM. This effect can be explained by the multiple small steps we take to generate the adversarial noise, instead of the one large step we take when using FGSM.

2.4 Adversarial Training

Goodfellow et al. [5] described that training a model with an adversarial objective function results into regularization, meaning that such a trained model performs well on both natural and adversarial data. They refer to this training method as adversarial training. The following objective function is proposed:

$$\tilde{J}(h_\theta(x), y) = \alpha J(h_\theta(x), y) + (1 - \alpha)(h_\theta(x + \delta), y). \tag{2.10}$$

It is shown that models trained with this objective function, where the perturbation, δ, is based on the fast gradient sign method, reduces the error on images perturbed using the same adversarial attack. On the downside, training with such an objective function does not offer protection against the PGD attack.

PGD proves to be more useful in adversarial training than FGSM. Models trained to be robust against PGD adversaries are robust against a wide range of first-order adversaries [18]. PGD generalizes well on both natural and perturbed data. This implies that we can choose $\alpha = 0$ in Equation

2.10, resulting in a objective function based on only the perturbed data. It is important to note that the capacity of the network should be large enough, as a small capacity results in the model sacrificing performance on natural examples to learn from the adversarial examples.

In adversarial training, the goal of the adversarial attack is to maximize the loss, while the goal of the model is to minimize the loss. This gives us the following saddle point problem, which an ideal robust classifier should achieve:

$$\min_{\theta} \max_{\delta} J(h_\theta(x+\delta), y), \tag{2.11}$$

such that $\|\delta\|_p \leq \epsilon$. For both inner, and outer optimization problems, we use the gradient of the loss function in solving the problem. A natural way to compute the gradient for the outer minimization problem is to replace the inputs of the model with their corresponding adversarial perturbations, and natural train the model on the perturbed data.

One important assumption that is made here is that Danskin's theorem holds even though the preconditions of the theorem do not hold. From Danskin's theorem follows that for continuously differentiable models, the outer minimization of Equation 2.11 can be solved by replacing the input of the model with its corresponding adversarial input and normally training the inner maximization. Madry et al. [18] suggest that the gradients can be used to solve Equation 2.11, despite the fact non-continuously models are used. Therefore, we make the same assumption in our experiments.

2.5 JPEG Compression

JPEG compression is a lossy compression format that exploits the fact that human perception is able to discriminate the brightness of images better than the color of images. Lossy implies that decompressed images might be different than the original images that have been compressed. The compression format is mainly used to reduce the size of images without making a visible difference. Research has shown that the compression format can also be used to offer protection against small adversarial perturbations [1][3]. The compression method consists of the following five steps:

1. **Color space transformation**
   The RGB image is transformed to YC_bC_r, where Y denotes the luminance component, and C_b and C_r the blue and red chromatic components respectively. This is done to isolate the luminance component, which humans can better discriminate than colors, from the color components.

2. **Downsampling**
   The chromatic channels of the YC_bC_r image are downsampled. This downsampling has little, to no effect on the perception of the image.

3. **Cosine transformation**
   The pixel values of the image are centered by subtracting 128 from its original value. The resulting pixels are put in 8×8 blocks. Each of the blocks is transformed using a discrete cosine transformation [20].

4. **Quantization**
   The resulting matrices from step 3 are quantized using a quantization matrix, Q. This is done by the following computation:

$$B_{i,j} = \text{round}\left(\frac{G_{i,j}}{Q_{i,j}}\right) \quad \forall i, j \in \{0, 1, \ldots, 7\}, \tag{2.12}$$

   where i denotes the horizontal coordinate, j the vertical coordinate, G a matrix obtained in the previous step, and B the quantized matrix. The degree of compression the user chooses determines the values of Q. A larger degree of compression implies larger values in the quantization matrix, and therefore greater compression.

5. **Huffman encoding**
   The components of B are arranged in a single vector by going through B in a zigzag pattern. The resulting vector is shrunk down using Huffman encoding [8].

The main advantage of using JPEG compression as a defense against adversarial attacks is that it is computationally inexpensive, especially compared to PGD-style adversarial training. On the downside, JPEG compression offers little to no protection against larger, still barely visible, adversarial perturbations [3]. For examples of an image compressed using different degrees of compression, see Figures A.5f, A.5g, and A.5h of the Appendix.

2.6 Unlearnable Examples

The research in protecting personal data from being exploited for the purpose of training neural networks is sparse. Shan et al. [22] made one of the first attempts to protect personal data against unauthorized data collection by designing a system called Fawkes. One year later, Huang et al. [7] proposed a method with the same objective as the Fawkes system. The method uses so-called error-minimizing noise to perturb the training data. The objective of this noise is to keep the loss on the test and validation data during the training of a model high while making the loss on the training data drop exceptionally fast. This results in the model thinking there is nothing left to learn from the training data while performing poorly on the validation and test data.

2.6.1 Generating Error-Minimizing Noise

To generate error-minimizing noise δ, we solve the following min-min problem:

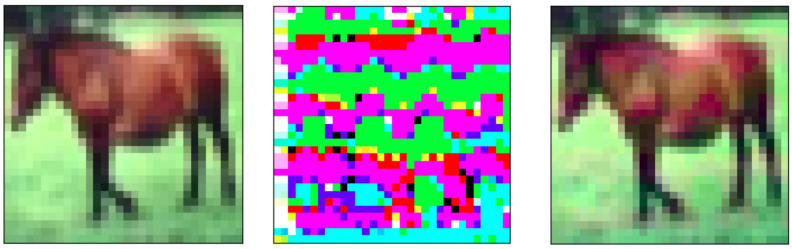
$$\min_{\theta} \min_{\delta} J(h_\theta(x+\delta), y), \tag{2.13}$$

such that $\|\delta\|_p \leq \epsilon$. The inner minimization concerns optimizing the error-minimizing noise. The outer minimization concerns finding the model parameters that minimize the loss. To solve this min-min problem, we first optimize θ for M steps, after which we optimize δ for T steps. The alternation of model training and noise optimization is repeated until the model achieves an accuracy lower than a specified value, λ, on the perturbed images.

The inner minimization is solved by altering the definition of Equation 2.9. Instead of setting a step in the direction the gradient of the loss function with respect to the input, we set a step in the negative direction. This way we minimize the loss of the perturbed samples, which results in the extremely fast drop of loss on the perturbed data during the training of a model. Generating error-minimizing noise is an iterative process, the noise at iteration $t+1$ is generated as follows:

$$\delta^{t+1} = \mathcal{P}(\delta^t - \alpha \operatorname{sign}(\nabla_x J(h_\theta(x+\delta^t), y))). \tag{2.14}$$

Two different types of error-minimizing noise are proposed: class-wise and sample-wise noise. Both use Equation 2.14 to generate the perturbation. To generate sample-wise noise, we directly use Equation 2.14 on each sample separately during the noise optimization step. An example of a CIFAR-10 image with sample-wise error-minimizing noise, generated using $\epsilon = \frac{8}{255}$ can be seen in Figure 2.4



(a) Natural (b) Noise (c) Perturbed image

Figure 2.4: Sample-wise noise bounded by $\|\delta\|_\infty \leq \frac{8}{255}$, applied to a CIFAR-10 image. Figure 2.4a shows the original image; Figure 2.4b the noise that is applied; Figure 2.4c the CIFAR-10 image with the added noise. Note that in order to visualize the noise in Figure 2.4b, it is multiplied by 250.

Class-wise noise is computed as a cumulative perturbation on all examples in the same class. This is done by adding all the noise masks belonging

to the same class to each other, after the noise optimization step. For an image in class k, the perturbation is generated by starting Equation 2.14 from δ_k. The pseudo-code for both class-wise and sample-wise noise generation can be found in Algorithm 1 of the Appendix.

2.6.2 Knowledge Gap

The error-minimizing noise method is a great step in the direction of making personal data unlearnable. However, Huang et al. [7] also mention some issues with the method. Our main concerns related to the proposed method are the resistance of the error-minimizing noise against adversarial training and data augmentation. Huang et al. [7] show that the error-minimizing noise is fairly resistant to four different data augmentation techniques, Cutout [2], Mixup [27], Cutmix [26], and Fast AutoAugment [15]. The highest accuracy achieved on CIFAR-10 [10] using a ResNet18 model and the previously listed data augmentation techniques is 58.51%, which is quite low compared to the reachable accuracy of 94.95% on clean data, as obtained by Huang et al. [7]. Interestingly, only these four data augmentation techniques are mentioned in the experiments. For non of the four data augmentation techniques, different parameters are explored. This lack of exploration combined with the general effects of data augmentation motivate us to test the resistance of the method against other data augmentation techniques.

The resistance of the error-minimizing noise against adversarial training as mentioned by Huang et al. [7] is also questionable. In the experiments the error-minimizing noise is bounded by $\delta_\infty \leq \frac{24}{255}$, while the attack used during the adversarial training is bounded by $\delta_\infty \leq \frac{8}{255}$. First, such large error-minimizing perturbations become obviously visible in the images, which breaks their requirement that the noise should be invisible. Second, even which such large error-minimizing perturbations, a PGD adversarial trained ResNet18 model can achieve an accuracy of 85%, as mentioned by Huang et al. [7]. This is a similar performance of a model adversarial trained on clean CIFAR-10 data. Therefore, we are motivated to further explore the resistance of error-minimizing noise against adversarial training.

Chapter 3

Research

We aim to improve the performance of models trained on error-minimizing perturbed datasets. This will give us a direction on how to strengthen the error-minimizing noise. In this chapter, we describe the experiments we conduct in order do this. We start by describing our threat model, stating the assumptions that we have made. These assumptions form the backbone of any claims that are made in this chapter. We then describe our approach, giving an overview of our experiments and the motivations behind them. After this, we describe the experiments we conduct to further explore the resistance of error-minimizing noise against adversarial training and data augmentation. The code used for all experiments can be found at GitHub[1]. The results of all experiments are discussed in the corresponding sections.

3.1 Threat Model

To be able to make any claims during the discussion of the experiments, we first describe the threat model. The threat model contains any assumptions that we have made, and therefore is an important foundation of claims made in the remainder of this chapter. We make the distinction between a defender and an attacker.

Defender

The objective of the defender is to generate noise in order to protect his or her personal images from being used in training neural networks without consent. The noise has two requirements:

1. The generated noise must be non-suspicious. This implies that it does not affect the normal usage of the images. For example, a defender does not want the noise to be obviously present on his or her Facebook

[1] https://github.com/TijnBerns/unlearnable-examples

profile picture. To ensure to noise is non-suspicious we use the L^∞-norm to bound the noise. We bound the noise the defender generates, δ, by $\|\delta\|_\infty \leq \frac{8}{255}$. This is equivalent to the bound mentioned by Huang et al. [7] to generate non-suspicious noise.

2. When training a model on the perturbed images, the model must not be able to learn to predict the classes corresponding to clean images. This means that the performance of a model trained on perturbed images is comparable to random guessing.

In reality, it would be ideal if the defender is able to generate noise using a different dataset. However, this requires dataset selection such that the classes of the dataset match with the classes of the images of the defender. Because this is a challenging task in itself, we limit the defender by only being able to perturb and upload entire datasets.

The defender has no knowledge of the training method or augmentations the attacker uses and can only create an unlearnable dataset once. If the images are uploaded to the internet, an attacker has access to it, and the defender is not able to make changes in the images anymore. Furthermore, we assume the defender only uses the L^∞-norm to generate the perturbations.

Attacker

The goal of the attacker is to train a model on an unlearnable dataset. We assume the attacker only has access to datasets of which all samples are perturbed since Huang et al. [7] already showed that when only a small portion is perturbed, models are able to learn from the data. The attacker only uses datasets of which each data sample is made unlearnable to train a model and is not able to generate error-minimizing noise.

When applying adversarial training, we limit the attacker by using perturbations, δ, such that $\|\delta\|_\infty \leq \frac{8}{255}$. We assume the attacker knows the L^∞-norm is used to generate the perturbations and knows whether sample-wise or class-wise noise is added to the images. The attacker does not know the method or model that is used to generate the error-minimizing noise masks, and therefore cannot replicate the noise masks.

3.2 Approach

We aim to improve the performance of neural networks trained on data with added error-minimizing noise. Ideally, a model trained on error-minimizing perturbed data achieves a performance similar to a model trained on clean data. By achieving this, we give a possible direction of improvement of the error-minimizing noise. In this section, we describe our approach to achieve this objective.

We focus on the following two aspects of the error-minimizing noise: its resistance against adversarial training, and its resistance against data augmentation. We hypothesize the opposite objectives of these methods compensate the effects of error-minimizing noise and therefore are able to improve the performance of models trained on error-minimizing perturbed data.

Before conducting experiments concerning our two identified aspects, we verify the effectiveness of adversarial training and adversarial attacks on clean data. These principles are closely related to error-minimizing noise for two reasons. First, for both adversarial training and the generation of error-minimizing noise, a similar bi-level optimization problem has to be solved. Second, to generate error-minimizing noise we use an adapted form of PGD. Therefore, the effectiveness of adversarial training and adversarial attacks must be verified. To do this, we adversarially train using the fast gradient sign method and projected gradient descent on an MNIST dataset [12] and a CIFAR-10 dataset [10]. This allows us to identify the advantages and disadvantages of both adversarial training techniques. Also, we can use the results as a comparison for adversarial training on error-minimizing perturbed data.

Our next step is to verify that error-minimizing noise deteriorates the training process of a neural network. This verification is important because without it, answering our research question might not be necessary. We verify this by generating class-wise and sample-wise error-minimizing noise for the CIFAR-10 dataset, using different sized L-balls. We then, train a ResNet18 model on the resulting perturbed datasets.

After verifying the effectiveness of adversarial attacks, adversarial training, and error-minimizing noise, we are able to conduct experiments concerning the error-minimizing noise. As previously mentioned, we hypothesize that adversarial training and data augmentation are both able to improve the performance of models trained on error-minimizing perturbed data. To test the effect of adversarial training, A ResNet18 model is adversarially trained on the perturbed datasets we obtained in the previous step. We train using both FGSM and PGD. For both attacks we use two different values for our perturbation size, $\epsilon = \frac{4}{255}$ and $\epsilon = \frac{8}{255}$.

To test the resistance of the error-minimizing noise against data augmentation, we train on perturbed data sets using a variety of data augmentation techniques. The augmentation techniques are chosen by looking at a fraction of all the noise masks. We identify two factors of the error-minimizing noise: color and texture. Different data augmentation techniques are used to break either of the two factors. The importance of the color factor is tested by removing color from the perturbed images. The importance of the texture is tested by applying JPEG compression, Gaussian noise, median blur, or course dropout. If a data augmentation technique improves the training of a ResNet18 model trained on error-minimizing perturbed data,

without sacrificing much performance of a ResNet18 model trained on clean data, we explore the data augmentation technique further. This exploration is done by testing the effect of combining the data augmentation technique with a grayscale transformation. We do not further experiment with less promising data augmentation techniques.

Finally, we adapt our treat model by assuming the defender knows that the attacker uses a grayscale transformation during training. This allows us to identify whether the extra information helps generating error-minimizing noise masks which are more resistant against a grayscale transformation. We generate error-minimizing noise masks on a grayscale transformed dataset and show that the resulting noise masks are dependent on the presence of color. To further improve the performance of models trained on the error-minimizing perturbed datasets, we use the data augmentations which showed to be most effective at improving the learning process in our previous experiments.

3.3 Adversarial Training

In this section, we describe the experiments that are conducted in order to test the effect of adversarial training on the accuracy of deep neural networks. We test the accuracy and loss of both regular and adversarial trained networks on the CIFAR-10 [10] and the MNIST [12] data set. We feed the trained models either clean or perturbed data. The perturbations are done using either PGD or FGSM. For both datasets, the parameters of the models we train are updated using stochastic gradient descent with an initial learning rate of 0.1. We update the learning rate using a cosine annealing scheduler.

3.3.1 MNIST

For the MNIST data set, we train a model consisting of two convolutional layers with 32 and 64 filters respectively, and a fully connected layer of size 1024. Both convolutional layers are followed by a max-pooling layer. A ReLu operation is applied after each of the convolutional layers and the fully connected layer.

We train the model in three different manners: without perturbing the data, by perturbing data using FGSM, and by perturbing data using PGD. During the training of the model using FGSM or PGD, all images are perturbed using the specific method before they are forwarded through the model. For each training manner, we train the model for 20 epochs. We save the model which achieves the highest average accuracy of a clean validation set, and a validation set that is perturbed during the evaluation of the model. The perturbation during the evaluation is done using the same adversarial attack as we are training the model with.

For both FGSM and PGD, ϵ is set to 0.3. PGD examples are generated using a random initial perturbation δ^0, followed by ten iterations of PGD, with a step size of 0.01. The average loss and average error of the different trained models on the test set are shown in Table 3.1.

	Regularly Trained		FGSM Trained		PGD Trained	
Attack	Average loss	Average error	Average loss	Average error	Average loss	Average error
None	0.01906	0.65%	0.24363	8.44%	0.02886	1.00%
FGSM	8.94725	96.61%	0.00447	0.13%	0.12731	4.33%
PGD	13.54142	100.00%	5.58916	91.85%	0.15590	5.05%

Table 3.1: Test results of MNIST models trained in a regular manner, by using PGD, or by using FSGM perturbed data.

We can see that both adversarial attacks work well on the regular trained model. When no attack is used the model achieves an error of nearly 0% on the test data. The error raises to 96.61% in case the data is perturbed using FGSM and even 1000% in case PGD is used.

The model trained using FGSM performs worse on clean MNIST data, while reaching a nearly perfect accuracy on FGSM perturbed data. This means that the model is overfitting on the FGSM perturbed data. The accuracy on the PGD perturbed data slightly drops, compared to the regular trained model, however, this accuracy is still extremely low.

The model trained using PGD performs well on clean data, and perturbed data using either FGSM or PGD. The conclusion we can draw from this is that in order to train a model to be robust against both FGSM and PGD, while also achieving high accuracy on clean data, it is better to train it using PGD than to train it using FGSM.

3.3.2 CIFAR-10

For the CIFAR-10 dataset, we train a ResNet18 model for 200 epochs. We regularly train the model or train the model using an adversarial objective function. During the training, we augment the data using a random crop, followed by a random horizontal flip.

In the case where we use an adversarial objective function, we train the model using the same perturbation methods as we used in the training of the previously described convolutional model. For both FGSM and PGD, ϵ is set to $8/255$. For PGD, we start from a random initial perturbation, δ^0, which is followed by 20 steps of PGD, with a step size of $\epsilon/10$. The average loss and average error of the different trained ResNet18 models are shown in Table 3.2.

Just like for the MNIST dataset, we can see that both adversarial attacks are effective. However, the difference between the two becomes more clear.

	Regular Trained		FGSM Trained		PGD Trained	
Attack	Average loss	Average error	Average loss	Average error	Average loss	Average error
None	0.18333	4.83%	0.44405	9.70%	0.53570	16.62%
FGSM	3.75599	62.79%	0.93117	18.30%	1.92110	48.33%
PGD	15.31369	100.00%	26.39517	98.91%	2.39671	55.52%

Table 3.2: Test results of Resnet18 models trained in a regular manner, by using PGD, or by using FSGM perturbed data.

When looking at the regularly trained model, we can see that projected gradient descent is much more effective than the fast gradient sign method, it achieves an error of 100% compared to 62.79% FGSM achieves.

The model trained using FGSM performs well on both clean and FGSM perturbed data, although we see that some accuracy is sacrificed training the model to be robust against FGSM. The model is not robust against PGD perturbed data at all. The error of the FGSM trained model on PGD perturbed data is still close to 100%.

More accuracy on clean data is sacrificed when the model is trained using PGD. The accuracy on FGSM perturbed data drops as well, compared to the FGSM trained model. However, the accuracy on PGD perturbed data is much higher compared to both regularly and FGSM trained models. From the results, we draw the following two conclusions. First, PGD is a much stronger adversarial attack than FGSM. Second, training using PGD results in more generalization on adversarial generated examples than training using FGSM. However, the cost of this is that more accuracy on clean data samples is sacrificed.

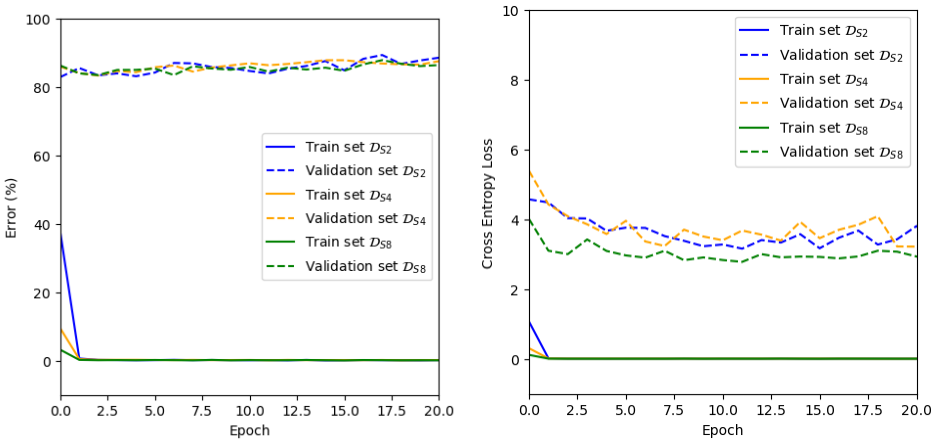
3.4 Unlearnable Examples

In this section, we first describe the experiments we conduct in order to test the effectiveness of the error-minimizing noise. We then describe the different experiments we conduct in order to test the effects of data augmentation and adversarial training on data perturbed using error-minimizing noise.

To get an overview of the strength of the error-minimizing perturbations, we generate both sample-wise and class-wise error-minimizing noise bounded by $\|\delta\|_\infty \leq \epsilon$, with $\epsilon \in \{\frac{2}{255}, \frac{4}{255}, \frac{8}{255}\}$. The exact pipeline for the noise generation can be found in Section A.2. The perturbation masks are generated on the CIFAR-10 dataset using data augmentation in the form of a random crop followed by a random horizontal flip. The resulting masks are applied to the CIFAR-10 dataset, giving us six different datasets (three sample-wise perturbed datasets, and three class-wise perturbed datasets). We use $\mathcal{D}_{S_i}$ and $\mathcal{D}_{C_i}$, to denote the sample-wise and class-wise perturbed

datasets respectively, where the noise is bounded by $\|\delta\|_\infty \leq \frac{i}{255}$. These datasets are referred to as "unlearnable datasets".

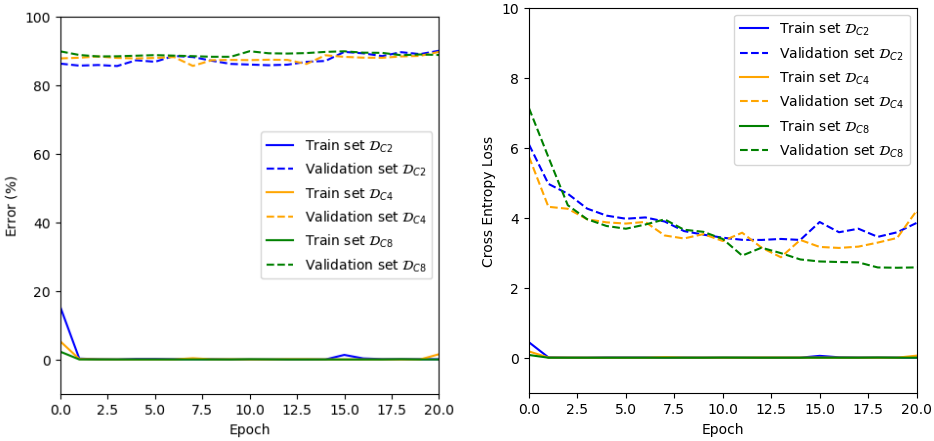
In order to test the effectiveness of the noise of each of the unlearnable datasets, we train a ResNet18 model for 200 epochs. The exact pipeline for the training process can be found in Section A.1. We do not use any form of data augmentation on the unlearnable datasets. The error and loss on both the perturbed training set and the clean validation set during the first 20 epochs of the training on the sample-wise and class-wise perturbed datasets, can be seen in Figure 3.1 and Figure 3.2 respectively. Most change in error and loss occurs in the first 20 epochs, for the complete training curves on the sample-wise perturbed datasets, refer to Figure A.1 of the Appendix. The error on a clean test set of the trained models is reported in the rightmost column of Table 3.3.



(a) Sample-wise error

(b) Sample-wise loss

Figure 3.1: Error on the validation set and the training set during the first 20 epochs of training a ResNet18 model on sample-wise unlearnable CIFAR-10 datasets.



(a) Class-wise error (b) Class-wise loss

Figure 3.2: Error on the validation set and the training set during the first 20 epochs of training a ResNet18 model on class-wise unlearnable CIFAR-10 datasets.

In Figure 3.1a and Figure 3.2a we can see that for all six unlearnable CIFAR-10 datasets, the training error drops to approximately zero after the second epoch, while the error on the clean validation set does not drop at all. For each unlearnable dataset, the error on the clean validation set stays around 82%, which makes the performance of the model almost as bad as random guessing. A similar phenomenon can be seen in Figures 3.1b and 3.2b. The conclusion we draw from this is that when no augmentation is used on the perturbed data set, even extremely small error-minimizing perturbations, bounded by $\|\delta\|_\infty \leq \frac{2}{255}$, are strong enough to drop the training error/loss exceptionally fast, making it impossible for the model to update its parameters properly.

We further verify the effectiveness of error-minimizing noise by applying basic data augmentation during the training process. The data augmentation consists of cropping and horizontally flipping the images. We train a ResNet18 model using the same training pipeline as our previous experiment. The test errors of the trained models on a clean test set are reported in Table 3.3

As expected, the performance of the model is better when we apply data augmentation. We can see that the augmentation results in more improvement when the model is trained on sample-wise perturbed data, than when it is trained on class-wise perturbed data. The results also verify that the class-wise noise is more effective as it results in worse performance than sample-wise noise. These results are used as a comparison for our following experiments.

Dataset	Augmentation	
	Yes	No
Clean CIFAR-10	4.83%	11.66%
$\mathcal{D}_{S_2}$	18.72%	81.16%
$\mathcal{D}_{S_4}$	74.58%	83.74%
$\mathcal{D}_{S_8}$	75.78%	83.70%
$\mathcal{D}_{C_2}$	21.10%	86.38%
$\mathcal{D}_{C_4}$	81.14%	86.56%
$\mathcal{D}_{C_8}$	84.98%	86.72%

Table 3.3: Error on a clean CIFAR-10 test set of ResNet18 models trained on different unlearnable CIFAR-10 datasets using either data augmentation in the form of a random crop followed by a random horizontal flip, or no data augmentation.

3.4.1 Resistance Against Adversarial Training

In the previous section we have seen that the effect of unlearnable noise is based on dropping the training loss and error exceptionally fast. To be able to learn from an unlearnable dataset, we have to somehow increase the loss on the training set, without sacrificing much accuracy on the clean validation and test set. Our first suggestion to achieve this is by adversarial training. The objective of an adversarial example is to maximize the loss with respect to the input. We hypothesize that the effect of error-minimizing noise is counteracted by the opposite objective of the adversarial attack used during training. In order to test this we adversarially train a ResNet18 model with the same training setting as in Section 3.3.2 on the unalienable CIFAR-10 datasets. During the training, we use a random crop followed by a random horizontal flip to augment the data. We adversarially train the model using both FGSM and PGD, where the attacks are bounded by $\|\delta\|_\infty \leq \frac{4}{255}$ or $\|\delta\|_\infty \leq \frac{8}{255}$. The error on the clean CIFAR-10 test set the trained models achieved are reported in Table 3.4. The best performances of models trained on each of the datasets are highlighted.

We observe that PGD-style adversarial training on any of the unlearnable datasets, except for the model trained with $\epsilon = \frac{4}{255}$ on $\mathcal{D}_{C_8}$, results in a similar performance to a model adversarial trained on clean data. Therefore, we argue that PGD-style training fully counteracts the effects of error-minimizing noise if the PGD perturbations are bounded by the same L-ball as the error-minimizing perturbations. In all cases, except for the model trained on $\mathcal{D}_{C_8}$, PGD-style training with $\epsilon = \frac{4}{255}$ results in better performance, than PGD-style straining with $\epsilon = \frac{8}{255}$. This can be explained, by our previous observation that training using larger adversarial perturbations

Dataset	FGSM Trained		PGD Trained	
	$\epsilon = 4/255$	$\epsilon = 8/255$	$\epsilon = 4/255$	$\epsilon = 8/255$
Clean CIFAR-10	11.62%	**9.70%**	12.56%	16.62%
$\mathcal{D}_{S_2}$	**12.10%**	16.96%	12.42%	18.46%
$\mathcal{D}_{S_4}$	**10.84%**	32.68%	12.18%	19.78%
$\mathcal{D}_{S_8}$	44.52%	35.54%	**15.70%**	18.82%
$\mathcal{D}_{C_2}$	**11.98%**	23.50%	12.26%	18.76%
$\mathcal{D}_{C_4}$	22.88%	22.82%	**12.01%**	18.60%
$\mathcal{D}_{C_8}$	46.40%	38.18%	66.82%	**17.92%**

Table 3.4: Error on a clean CIFAR-10 test set of ResNet18 models adversarially trained using either PGD or FGSM on different unlearnable CIFAR-10 datasets.

results in more performance being sacrificed on the clean test set.

In case FGSM-style training is used, we observe that the performance is better than PGD-style adversarial trained models in case of small error-minimizing perturbations. However, in general, the performance of the PGD-style trained models is better than, or similar to the performance of the FGSM-style trained models. This is not unexpected, as a model trained to be robust against PGD adversaries, is resistant to a wide variety of adversarial attacks [18]. We argue that PGD-style adversarial training is the better and more reliable method out of the two at improving the performance of models trained on unlearnable datasets.

An interesting case to mention is the model trained using FGSM with $\epsilon = \frac{4}{255}$ on $\mathcal{D}_{S_4}$. We observe that training on this unlearnable dataset using the specific training method results in better performance than when training using the same training method on a clean dataset. This is unexpected because adversarial training on unlearnable datasets results in two layers of noise on top of the original images of a dataset, the adversarial noise, and the error-minimizing noise. In case of adversarial training on clean data, there is only one layer of noise on top of the original images. Therefore, we expect the model trained on clean data to perform better than the model trained on $\mathcal{D}_{S_4}$. We do not further explore this phenomenon in our research.

3.4.2 Resistance Against Data Augmentation

In the previous section, we have seen that adversarial training improves the performance of a model trained on an unlearnable dataset. Although the results are promising, there is still a significant drop in accuracy compared to a model trained in a standard manner on a clean dataset. In this sec-

tion, we will explore the effects of applying a variety of data augmentation methods on the unlearnable CIFAR-10 datasets. To determine which data augmentations are potentially effective we plot the first three noise masks of the sample-wise generated masks in Figure 3.3. More examples of sample-wise noise masks and examples of class-wise noise masks can be found in Figure A.2 and Figure A.3 of the Appendix.

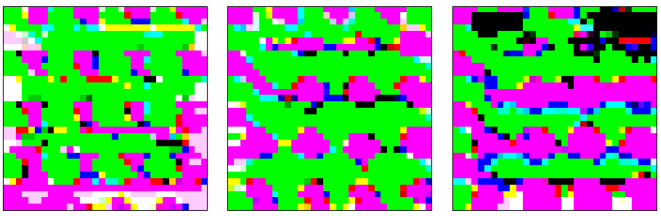


Figure 3.3: The first three sample-wise noise masks of the unlearnable CIAFR10 dataset. Noise is bounded by $\|\delta\|_\infty \leq \frac{8}{255}$. In order to visualize the noise, the masks have been multiplied by a factor of 255.

In these figures we can see that there are at least two clear patterns in the noise masks. First, magenta and green are more prominent colors in the noise masks than others. Second, there is often a dot or line pattern in the masks, where magenta and green are alternated. The *RGB* values of magenta and green are $(255, 0, 255)$ and $(0, 255, 0)$, which makes the colors opposite from one another. We hypothesize that the opposite colors and the alternation of the two colors in the patterns result in the textures being easy for a model to detect. Therefore, a model trained on the perturbed data focuses on learning the texture of the noise instead of learning the texture of the original images.

We assume that both color and texture are an important factor of the error-minimizing noise. To test the importance of color, we transform the unlearnable datasets using a grayscale transformation. To test the importance of texture, we use the following data augmentation techniques in order to distort it: JPEG compression, Gaussian noise, median blur, and course dropout. In the remainder of this section we discuss the experiments we conduct regarding these data augmentations in detail.

Grayscaling

Previous research has shown that color is not a critical feature for accurate classification of images [4][25]. This, and our observation that there are prominent colors in the noise masks motivates us to transform the images of the unlearnable datasets into grayscale images. Before describing our experiments, let us give some more intuition why grayscaling the unlearnable datasets would have a positive effect on the performance of models trained

on the resulting gray datasets. To transform an image into a grayscale image, the weighted average of the RGB channels is computed as follows:

$$Y = \text{round}\left(0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B\right), \tag{3.1}$$

this computation is done for each pixel separately. The resulting value is used for all RGB channels, removing the color from the image.

We compute the weighted average using Equation 3.1 for magenta and green. This results in $Y_m = 105$ for magenta, and $Y_g = 150$ for green. The difference between these two values is much less than the average difference between the RGB values of magenta and green (45 vs. 255). Therefore, in the colored perturbed images the textures of the noise are much more noticeable than in the grayscale transformed images. We hypothesize that due to the smaller difference in color values, and our observation that the magenta and green are often alternated in the noise masks, the textures in the noise masks become harder for a model to detect. This results in a model focusing less on learning the noise textures, and more on learning the actual images. Therefore, a model will perform better when it is trained on grayscale unlearnable datasets, than when it is trained on colored unlearnable datasets.

To test our hypothesis, we transform all images in the unlearnable CIFAR-10 datasets into grayscale images. We train a ResNet18 model on the resulting datasets according to the training pipeline described in Section A.1. To further improve the training, we augment the data by cropping and horizontally flipping the images. Errors on the colored clean test set of the ResNet18 model trained on the different unlearnable gray CIFAR-10 datasets are reported in Table 3.5.

Dataset	Test error
Clean CIFAR-10	7.10%
$\mathcal{D}_{S_2}$	6.88%
$\mathcal{D}_{S_4}$	6.62%
$\mathcal{D}_{S_8}$	9.60%
$\mathcal{D}_{C_2}$	11.08%
$\mathcal{D}_{C_4}$	24.66%
$\mathcal{D}_{C_8}$	37.84%

Table 3.5: Error on clean CIFAR-10 test set of a ResNet18 model trained on different unlearnable gray CIFAR-10 training sets and a clean CIFAR-10 dataset. During training, images are transformed using a grayscale transformation and augmented using a random crop and a random horizontal flip.

Comparing these results with the ones reported in Table 3.3, we observe that the grayscale transformation greatly improves the performance of models trained on the unlearnable datasets. The performances of models trained on sample-wise perturbed datasets are similar to that of the model trained on clean data. In case the model is trained on class wise perturbed data, we observe that the performance of the model quickly drops as the allowed perturbation size of the error-minimizing noise masks is increased.

From these results, we conclude that the colors of the noise masks are an important factor in making the data unlearnable. However, for large perturbations, the textures of the noise masks remain in the images after grayscaling them. These textures become more obvious as we increase the size of the perturbations. In case of class-wise noise, all images in the same class contain the same noise pattern. Therefore, increasing the size of the class-wise perturbations results in the model focusing more on learning the correlation between the textures in the noise and the labels, instead of learning the correlation between the underlying images and the labels. Which explains why performance weakens as the size of class-wise error-minimizing perturbations is increased.

JPEG compression

JPEG compression is known to protect against small adversarial perturbations [1][3]. This motivates us to apply JPEG compression on the unlearnable CIFAR-10 datasets. We compress both the clean CIFAR-10 dataset, and the unlearnable datasets using JPEG compression with the following degrees of compression: 40, 60, and 80. Examples of a compressed CIFAR-10 image using the previously mentioned degrees of compression, can be found in Figures A.5f, A.5g and A.5h of the Appendix. We train a ResNet18 model using the training pipeline described in Section A.1 on each of the 21 compressed datasets (three for each of the unlearnable datasets, and three for the clean CIFAR-10 dataset). During training we augment the data using a random horizontal flip and a random crop. The test errors of the ResNet18 model trained on each of the resulting datasets are summarized in Figure 3.4. These results should be compared to a test error of 4.83% a ResNet18 model is able to achieve when trained on clean CIFAR-10 data.

Compared to the results in Table 3.3, we observe that JPEG compression greatly improves the performance of models trained on unlearnable datasets. We see that as we increase the degree of compression, the error on the unlearnable datasets drops. However, the error on the clean datasets rises when increasing the compression. Since we want to achieve a performance comparable to a ResNet18 model trained on a clean CIFAR-10 dataset, we have to limit the degree of compression.

We argue that JPEG compression takes away the textures in the noise. We hypothesize that due to the positive effects of JPEG compression and

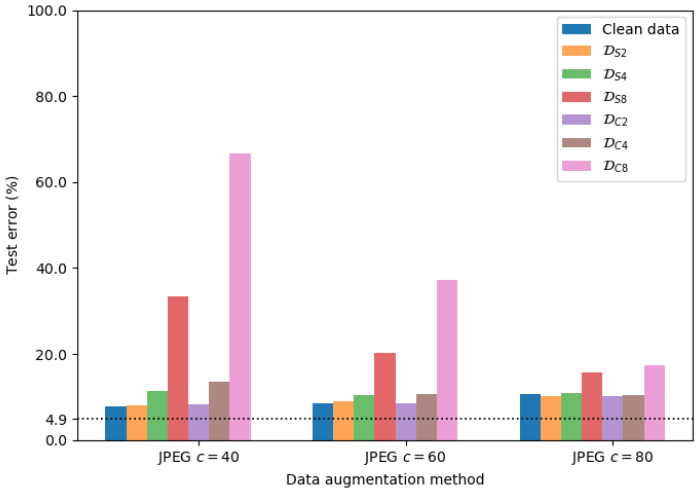


Figure 3.4: Test error of a Resnet18 model trained on different unlearnable CIFAR-10 datasets. Samples from the datasets have been compressed using JPEG compression with different degrees of compression. The accuracy a ResNet18 model trained on clean data is able to achieve is visualized by the dotted line.

grayscale transformation on the training performance, combining JPEG compression with grayscale transformation further improves the performance of models trained on the unlearnable datasets. We grayscale all 21 previously obtained JPEG compressed datasets, and train a ResNet18 model on the resulting 21 datasets. The test error of the trained model on each of the resulting datasets can be found in Figure 3.5.

Comparing these results with the ones in Figure 3.4 and Table 3.3, we observe that in case of large error-minimizing perturbations, combining grayscale transformation with JPEG compression results in better performance than when only one of the two methods is used. This effect the strongest for models trained on $\mathcal{D}_{S_8}$ and $\mathcal{D}_{C_8}$. In case a compression degree of 40 is used, the performances of all models, with the exception of the model trained on $\mathcal{D}_{C_8}$, are similar. To reduce the error of the model trained on $\mathcal{D}_{C_8}$ a larger degree of compression must be used. However, this results in more performance being sacrificed on both the clean dataset and the other unlearnable datasets. Although we are sacrificing performance by increasing the degree of compression, we are able to achieve better performances using these data augmentations than when using adversarial training. Therefore, we argue that a grayscale transformation combined with JPEG compression is an effective method to improve performance of models trained on unlearnable datasets.

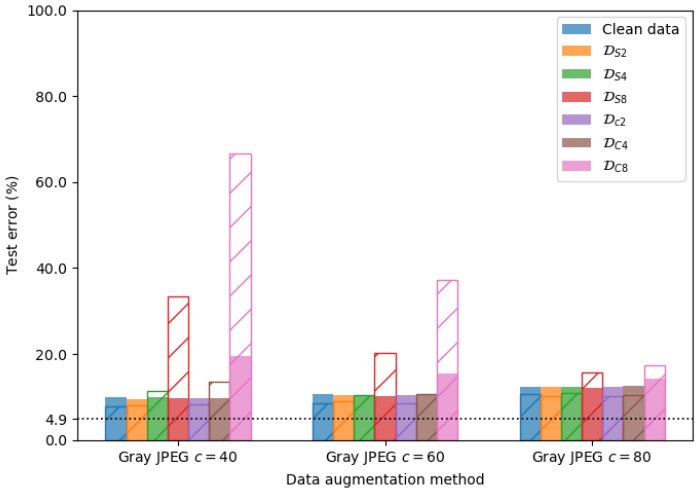


Figure 3.5: Test error of a ResNet18 model trained on different unlearnable CIFAR-10 datasets. Samples from the datasets have been compressed using JPEG compression with different degrees, and transformed into grayscale images. The accuracy a ResNet18 model trained on clean data is able to achieve is visualized by the dotted line. The results from Figure 3.4 are visualized by the dashed bars.

.

Gaussian Noise

We have seen that by using a combination of grayscale transformation and JPEG compression, we can remove the effect of unlearnable samples for a large part. In this section we apply Gaussian noise, changing the distribution of the unlearnable datasets.

We train a ResNet18 model according to the training pipeline described in Appendix A.1 on all six unlearnable datasets, and a clean CIFAR-10 dataset. During the training, we augment the data by adding Gaussian noise with zero mean and a standard deviation of 0.05, 0.1, or 0,2, to the data samples. Besides adding Gaussian noise, we apply a random horizontal flip and a random crop. The errors of the trained models on a clean test set are summarized in Figure 3.6.

Compared to the results in Table 3.3, we observe that adding Gaussian noise only slightly improves performance of models trained on unlearnable data. Adding Gaussian noise with a larger standard deviation results in better performance. This effect is the strongest for the models trained on $\mathcal{D}_{S_2}$ and $\mathcal{D}_{S_4}$. We argue that the error-minimizing perturbations are fairly resistant to Gaussian noise, especially considering the Gaussian noise we add to the samples is much larger than the error-minimizing perturbations. However, since there is still a noticeable difference compared to models trained

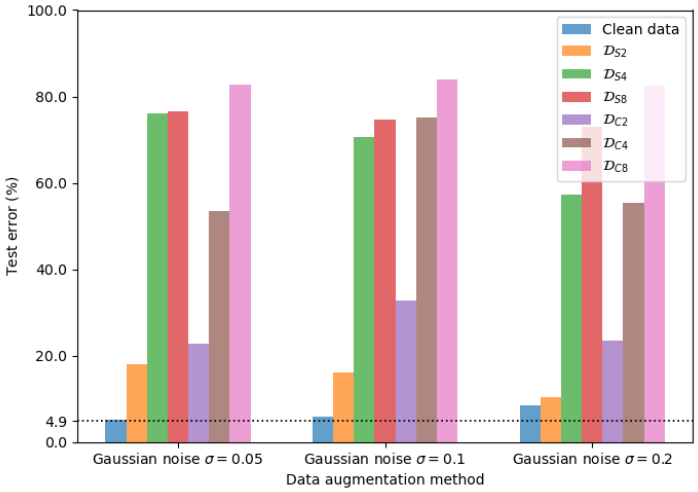


Figure 3.6: Test error of a ResNet18 model trained on different unlearnable CIFAR-10 datasets. Samples from the datasets are augmented during training by applying Gaussian noise. The accuracy a ResNet18 model trained on clean data is able to achieve is visualized by the dotted line.

on unlearnable datasets without adding noise, we will test the effect of transforming the images with added Gaussian noise into grayscale images. We train a ResNet18 model on the resulting gray images using the training pipeline as described in Appendix A.1. During the training we add Gaussian noise to the images and augment the data using a random crop followed by a random horizontal flip. The error of the ResNet18 on the different unlearnable datasets is summarized in Figure 3.7.

Comparing these results with the ones in Table 3.5 and Figure 3.6, we observe that using a combination grayscale transformation and Gaussian noise results in significantly better performance than when only one of the two methods is applied. In case Gaussian noise with $\sigma = 0.05$ is applied, the models trained on each of the unlearnable datasets, except for $\mathcal{D}_{C_8}$, are able to achieve a comparable performance to a model trained on clean data. To increase the performance of the model trained on $\mathcal{D}_{C_8}$, we need to apply Gaussian noise with a larger standard deviation. Unfortunately, by enlarging the standard deviation, performance of models trained on the other datasets is sacrificed. Although we sacrifice performance by enlarging the standard deviation, we argue that using the transformation is an effective method to counteract the effect of error-minimizing noise.

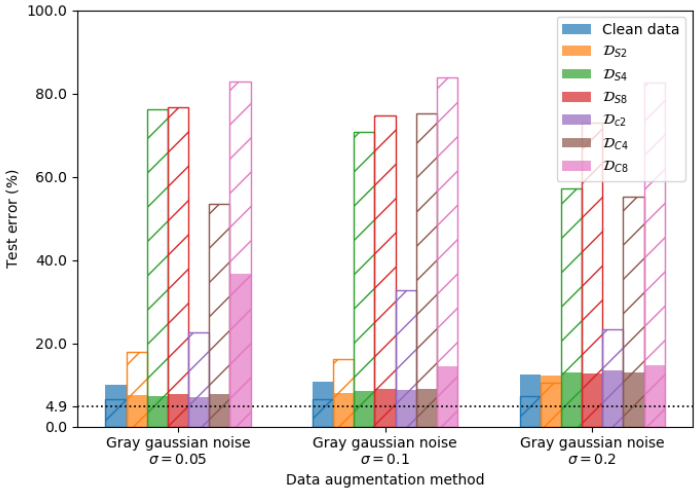


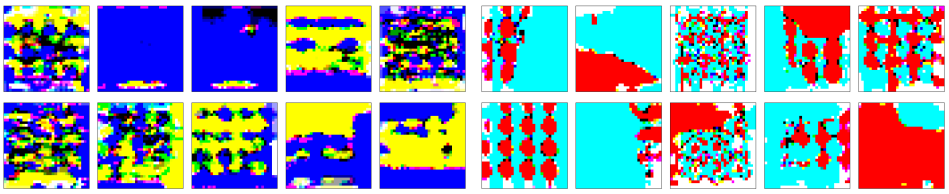
Figure 3.7: Test error of a ResNet18 model trained on different unlearnable CIFAR-10 datasets. Samples from the datasets are augmented during training by adding Gaussian noise, and transformed into grayscale images. The accuracy a ResNet18 model trained on clean data is able to achieve is visualized by the dotted line. The results from Figure 3.6 are visualized by dashed bars.

3.5 Adaptive Defender

In our previous experiments, we have shown that the effects of error-minimizing noise depend on the presence of color. We have seen that the performance of models trained on the unlearnable datasets improves by transforming the images into grayscale images, especially when the grayscale transformation is combined with Gaussian noise or JPEG compression. Therefore, our research question has already been answered. In this section, we look beyond our initial goal by adapting our threat model. We assume that the defender knows the attacker uses a grayscale transformation to improve the training on perturbed images. Other assumptions that are made in the threat model remain the same.

The most intuitive way to prevent an attacker applying grayscale transformation from learning from the error-minimizing perturbed data is to generate the error-minimizing noise masks on grayscale images. To simulate a defender with such knowledge, we generate sample-wise and class-wise error-minimizing noise using $\epsilon = \frac{8}{255}$ on a grayscale CIFAR-10 dataset according to the pipeline described in Section A.2. During the generation of the noise masks, the data is augmentation by cropping and horizontally flipping the images. The resulting noise masks are added to a colored CIFAR-10 dataset. We denote the sample-wise perturbed dataset by $\mathcal{D}^*_{S_8}$, and the class-wise per-

turbed dataset by $\mathcal{D}^*_{C_8}$. The first 10 noise masks of $\mathcal{D}^*_{S_8}$, and all noise masks of $\mathcal{D}^*_{C_8}$ can be found in Figure 3.8



(a) Sample-wise (b) Class-wise

Figure 3.8: Noise masks generated on a grayscale CIFAR-10 dataset visualized. Figure 3.8a shows the first 10 masks of $\mathcal{D}^*_{S_8}$; Figure 3.8b shows all 10 masks of $\mathcal{D}^*_{C_8}$. Note that in order to visualize the noise, it has been multiplied by a factor of 255.

Similar to the noise masks generated on a colored dataset, there are clear patterns and prominent colors in the masks. Note that, like we have seen with the colors of the masks in Figures A.2 and A.3, in both sets of noise masks the most prominent colors are opposite from each other. This strengthens our hypothesis made in Section 3.4.2, stating that the fact that the colors are opposite from each other makes it easier for the model to detect the noise patterns. We speculate that when we have no color information when generating the noise masks, the prominent colors in the mask will be two random colors opposite from each other.

3.5.1 Effectiveness

First, we need to test whether $\mathcal{D}^*_{S_8}$ and $\mathcal{D}^*_{C_8}$ are unlearnable when we apply data augmentation in the form of cropping and horizontally flipping. We do this by training a ResNet18 model on $\mathcal{D}^*_{S_8}$ and $\mathcal{D}^*_{C_8}$ according to the training pipeline described in Section A.1. The model trained on $\mathcal{D}^*_{S_8}$ achieved an error of 75.14%, the model trained on $\mathcal{D}^*_{C_8}$ an error of 82.40%. Therefore, both perturbed datasets are effective at deteriorating the training process of a neural network when basic data augmentation is used.

3.5.2 Resistance Against Grayscale Transformation

To test whether applying a grayscale transformation is an effective way of improving the performance of models trained on $\mathcal{D}^*_{S_8}$ and $\mathcal{D}^*_{C_8}$, we train a ResNet18 model using training pipeline A.1 on both datasets. During the training, we transform the images using grayscale transformation and augment the images using a random crop followed by a random horizontal flip. The model trained on $D^*_{S_8}$ achieves an error of 23.96% on the clean test set; the model trained on $\mathcal{D}^*_{C_8}$ achieves an error of 49.66%. The error of the models trained on $\mathcal{D}^*_{S_8}$ and $\mathcal{D}^*_{C_8}$ is higher than the error of the

models trained on $\mathcal{D}_{S_8}$ and $\mathcal{D}_{C_8}$. Therefore, we conclude that generating error-minimizing noise on a grayscale transformed dataset makes the noise more resistant against grayscale transformation during training. However, grayscale transformation is still an effective method to improve the training process.

The next step is to test whether a grayscale transformation combined with Gaussian noise or JPEG compression, is an effective method to improve the performance of models trained on $\mathcal{D}^*_{S_8}$ and $\mathcal{D}^*_{C_8}$. We do this by training a ResNet18 model for 200 epochs on both unlearnable datasets. The training process is according to the pipeline described in Section A.1. During the training, we transform the perturbed images into grayscale images. We then apply either Gaussian noise or perform JPEG compression. Which is followed by a random crop and a random horizontal flip. For JPEG compression we use the following degrees of compression: $c = 40$, $c = 60$, and $c = 80$. The Gaussian noise has zero mean with a standard deviation of 0.05, 0.10, or 0.20. The test errors of the trained models on a clean test set are reported in Table 3.6 and should be compared to the results in Figure 3.5 and Figure 3.7.

	JPEG Compression			Gaussian Noise		
Dataset	$c = 40$	$c = 60$	$c = 80$	$\sigma = 0.05$	$\sigma = 0.10$	$\sigma = 0.20$
Clean CIFAR-10	9.96%	10.76%	12.44%	7.36%	8.70%	13.04%
$\mathcal{D}^*_{S_8}$	19.38%	18.68%	20.01%	14.64%	14.84%	17.46%
$\mathcal{D}^*_{C_8}$	23.68%	17.76%	16.12%	43.46%	23.00%	16.98%

Table 3.6: Test error of ResNet18 model trained on different unlearnable datasets. During training data is augmented by applying JPEG compression or by adding Gaussian noise, both are followed by random cropping and random horizontally flipping

We observe that by training using the data augmentations the ResNet18 model is still able to learn from the unlearnable datasets. By adding Gaussian noise with a standard deviation of 0.05 to $D^*_{C_8}$, the model is able to achieve a test error of 43.46% on the clean test. For all data augmentation techniques used in this section, noise masks generated on grayscale images are more resistant to the data augmentations than the noise masks generated on colored images. However, the models are able to achieve a relatively good performance. We conclude that even when the defender knows that the attacker uses a grayscale transformation and adapts the noise generation based on this, the grayscale transformation still compromises the effects of the error-minimizing noise. Therefore, the error-minimizing noise as proposed by Huang et al. [7] should be adapted to make it less dependent on the presence of color.

Chapter 4

Conclusions

This research aimed to improve the performance of neural networks trained on data with added error-minimizing noise as proposed by Huang et al. [7]. By doing this we are able to show weak points of the error-minimizing noise and thereby give a possible direction of strengthening the noise. Our experiments focused on the resistance of the noise against adversarial training and data augmentation. We have shown that by using either of the two methods, a ResNet18 model trained on error-minimizing perturbed CIFAR-10 data is able to achieve a performance close to that of a model trained on clean CIFAR-10 data.

To test the effects of adversarial training on the error-minimizing perturbed data, we adversarially trained a ResNet18 model using PGD or FGSM on error-minimizing perturbed CIFAR-10 data. PGD-style adversarial training offers more robustness against larger error-minimizing perturbations. Using this training method we are able to achieve a performance similar to that of a model trained in the same manner on clean CIFAR-10 data. FGSM-style adversarial training training offers more robustness against smaller error-minimizing perturbations. Interestingly, in case FGSM-style adversarial training is applied and the error-minimizing perturbations are extremely small, the performance of the trained model is better than when the same training method is applied on a clean dataset. We did not further explore this phenomenon. Since adversarial training on error-minimizing perturbed data results in similar performance to adversarial training on clean data, we argue that the training method effectively removes error-minimizing perturbations.

To the effects of data augmentation on error-minimizing perturbed data, we identified two factors of the noise masks: color, and pattern. In our experiments, we tested the importance of each of the factors separately. To test the importance of the patterns in the noise masks, we applied JPEG compression, Gaussian noise, course dropout, and median blur. Out of these augmentation techniques, JPEG compression and Gaussian blur managed

to improve the performance of models trained on unlearnable data. To test the importance of color we transformed the perturbed data into grayscale images. We have shown that by removing the color from the unlearnable data, the performance of a model trained on the data greatly improves. Therefore, the effect of error-minimizing noise highly depends on the presence of color. Generating error-minimizing noise on grayscale image, does not make the noise more resistant against such transformations. In order to make the noise more resistant against grayscale transformations, the method of generating the noise must be adapted. Further, research is required for this.

Our previous described training method can be further improved upon by combining the idea of removing the color from the perturbed data, with breaking the patterns in the noise masks. By adding Gaussian noise and transforming the resulting images into grayscale images, we managed to reduce the error of a model trained on the class-wise perturbed data from 89.48% to 13.74%. Note that this is the strongest type of non-suspicious error-minimizing noise. Similar performance can be achieved by combining JPEG compression with applying a grayscale transformation. Although this performance is not as good as a model trained on clean data, we argue that using these data augmentations we are able to break the effect of the unlearnable noise.

The research showed that the error-minimizing noise proposed by Huang et al. [7] is not resistant against adversarial training and data augmentation. Applying grayscale transformation showed to be most effective, especially when combined with applying JPEG compression or adding Gaussian noise. Therefore, the noise highly depends on the presence of color. In order to make the error-minimizing noise more resistant against both data augmentation and adversarial training, further research is required.

Bibliography

[1] Nilaksh Das et al. "Keeping the Bad Guys Out: Protecting and Vaccinating Deep Learning with JPEG Compression". 2017. arXiv: 1705.02900.

[2] Terrance DeVries and Graham W. Taylor. "Improved Regularization of Convolutional Neural Networks with Cutout". 2017. arXiv: 1708.04552.

[3] Gintare Karolina Dziugaite, Zoubin Ghahramani, and Daniel M. Roy. "A study of the effect of JPG compression on adversarial images". 2016. arXiv: 1608.00853.

[4] Robert Geirhos et al. "ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness". In: *International Conference on Learning Representations*. 2019.

[5] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations*. 2015.

[6] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 770–778.

[7] Hanxun Huang et al. "Unlearnable Examples: Making Personal Data Unexploitable". In: *International Conference on Learning Representations*. 2021.

[8] David A. Huffman. "A Method for the Construction of Minimum-Redundancy Codes". In: *Proceedings of the IRE*. 1952, pp. 1098–1101.

[9] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *International Conference on Machine Learning*. 2015, pp. 448–456.

[10] Alex Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". 2012.

[11] Martha Larson et al. "Pixel Privacy: Increasing image appeal while blocking automatic inference of sensitive scene information". In: *Working Notes Proceedings of the MediaEval Workshop*. 2018.

[12] Yann LeCun and Corinna Cortes. “MNIST handwritten digit database”. 2010.

[13] Yann LeCun et al. “A theoretical framework for back-propagation”. In: *Connectionist models summer school*. 1988, pp. 21–28.

[14] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE*. 1998, pp. 2278–2324.

[15] Sungbin Lim et al. “Fast AutoAugment”. In: *Advances in Neural Information Processing Systems*. 2019.

[16] Zhuoran Liu, Zhengyu Zhao, and Martha Larson. “Pixel Privacy 2019: Protecting sensitive scene information in images”. In: *Working Notes Proceedings of the MediaEval Workshop*. 2019.

[17] Ilya Loshchilov and Frank Hutter. “SGDR: Stochastic Gradient Descent with Warm Restarts”. In: *International Conference on Learning Representations*. 2017.

[18] Aleksander Madry et al. “Towards Deep Learning Models Resistant to Adversarial Attacks”. In: *International Conference on Learning Representations*. 2019.

[19] Payman Mohassel and Yupeng Zhang. “SecureML: A System for Scalable Privacy-Preserving Machine Learning”. In: *IEEE Symposium on Security and Privacy*. 2017, pp. 19–38.

[20] Ahmed Nasir, T. Natarajan, and K.R. Rao. “A Method for the Construction of Minimum-Redundancy Codes”. In: *IEEE Transactions on Computers*. 1974, pp. 90–93.

[21] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision*. 2015, pp. 211–252.

[22] Shawn Shan et al. “Fawkes: Protecting Privacy against Unauthorized Deep Learning Models”. 2020. arXiv: `2002.08327`.

[23] Reza Shokri and Vitaly Shmatikov. “Privacy-Preserving Deep Learning”. In: *ACM SIGSAC Conference on Computer and Communications Security*. 2015, pp. 1310–1321.

[24] Christian Szegedy et al. “Intriguing properties of neural networks”. 2014. arXiv: `1312.6199`.

[25] Yiting Xie and David Richmond. “Pre-training on Grayscale ImageNet Improves Medical Image Classification”. In: *Proceedings of the European Conference on Computer Vision Workshops*. 2018.

[26] Sangdoo Yun et al. “CutMix: Regularization Strategy to Train Strong Classifiers With Localizable Features”. In: *CVF International Conference on Computer Vision*. 2019, pp. 6022–6031.

[27] Hongyi Zhang et al. “mixup: beyond empirical risk minimization”. In: *International Conference on Learning Representations*. 2018.

Appendix A

Error-minimizing noise

A.1 Training pipeline

For all of our experiments where we train a ResNet18 model, we train it for 200 epochs using stochastic gradient descent. The initial learning rate is 0.1, the momentum 0.9, and the weight decay 0.0005. A cosine annealing scheduler is used to update the learning rate during the training process. The cross-entropy loss is used to compute the loss of the model predictions and the true labels. In all cases, the model which performs best on the clean validation set, is used as the model to evaluate the performance on the test set.

When training on CIFAR-10 or perturbed CIFAR-10 data, we normalize the data by subtracting the mean and dividing by the standard deviation of the data. The mean we used is $(0.4914, 0.4822, 0.4465)$. The standard deviation we used is $(0.2023, 0.1994, 0.2010)$.

A.2 Pipeline for Generating Unlearnable Examples

The generation of error-minimizing noise masks follows Algorithm 1. We use 20 epochs of model optimization ($M = 20$), followed by 20 steps of perturbation optimization ($T = 20$) for each data sample in the training set. These two steps are repeated until the error during the evaluation step drops below 1% ($\lambda = 0.01$).

The model optimization is done using stochastic gradient descent with an initial learning rate of 0.1, a momentum of 0.9 and weight decay of 0.0005. The cross entropy loss is used to compute the loss of the predictions of the model. The optimization of the perturbations is done by performing 20 iterations of Equation 2.14 with a step size of $\epsilon/10$.

Algorithm 1: Generating error-minimizing noise masks

input : Model h_θ, Dataset $\mathcal{D} = (x, y)$, Bound ϵ, Stop error λ, Training steps M, PGD steps T

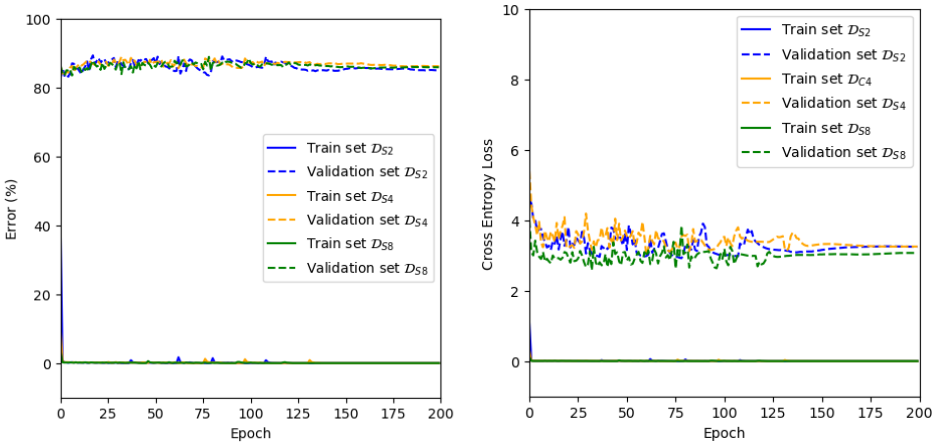
output: Error-minimizing masks δ for dataset $\mathcal{D}$

1 $\delta \leftarrow \text{Uniform}(-\epsilon, \epsilon)$
2 *error* $\leftarrow \infty$
3 **while** *error* $\geq \lambda$ **do**
4 **for** *train step* $\leftarrow 1$ **to** M **do**
5 $x_i, y_i \leftarrow \text{Next}(\mathcal{D})$ ▷ Get the next batch from $\mathcal{D}$
6 $\tilde{x_i} \leftarrow \text{AddNoise}(x_i, y_i, \delta)$
7 $h_\theta \leftarrow \text{Optimize}(h_\theta, \tilde{x_i}, y_i)$
8 **end**
9 **for** $(x_i, y_i) \in \mathcal{D}$ **do**
10 $\tilde{x_i} \leftarrow \text{AddNoise}(x_i, y_i, \delta)$
11 **for** *perturbation step* $\leftarrow 1$ **to** T **do**
12 $\eta \leftarrow \text{Perturbation}(h_\theta, \tilde{x_i}, y_i)$ ▷ Perform one step of Equation 2.14
13 $\eta \leftarrow \text{Clip}(\eta, -\epsilon, \epsilon)$
14 $\eta \leftarrow \text{Clip}(x_i + \eta, 0, 1) - x_i$
15 **end**
16 **if** *sample wise* **then**
17 $\delta_i \leftarrow \eta$
18 **end**
19 **if** *class wise* **then**
20 **for** $\eta_j \in \eta$ **do**
21 $c \leftarrow \text{Class}(\eta_j)$ ▷ Get the class corresponding to mask η_j
22 $\delta_c \leftarrow \text{Clip}(\delta_c + \eta_j, -\epsilon, \epsilon)$
23 **end**
24 **end**
25 **end**
26 *error* $\leftarrow \text{Eval}(h_\theta, x + \delta, y))$
27 **end**

The add noise method in lines 6 and 10, adds noise to a batch x_i. In case of generating sample-wise noise, the noise is added in a sample-wise manner. In case of generating class-wise noise, the noise is added according to the classes of the images in batch x_i.

A.3 Full Training Curve Unlearnable Data

The images in this section correspond to the experiments of Section 3.4. They show the loss and error during the entire training process of a ResNet18 model trained on sample-wise perturbed datasets according to the training pipeline described in Section A.1.



(a) Sample-wise error

(b) Sample-wise loss

Figure A.1: Error on the validation set and the training set during the entire training process of a ResNet18 model on sample-wise perturbed CIFAR-10 datasets.

A.4 Error-Minimizing Noise Examples

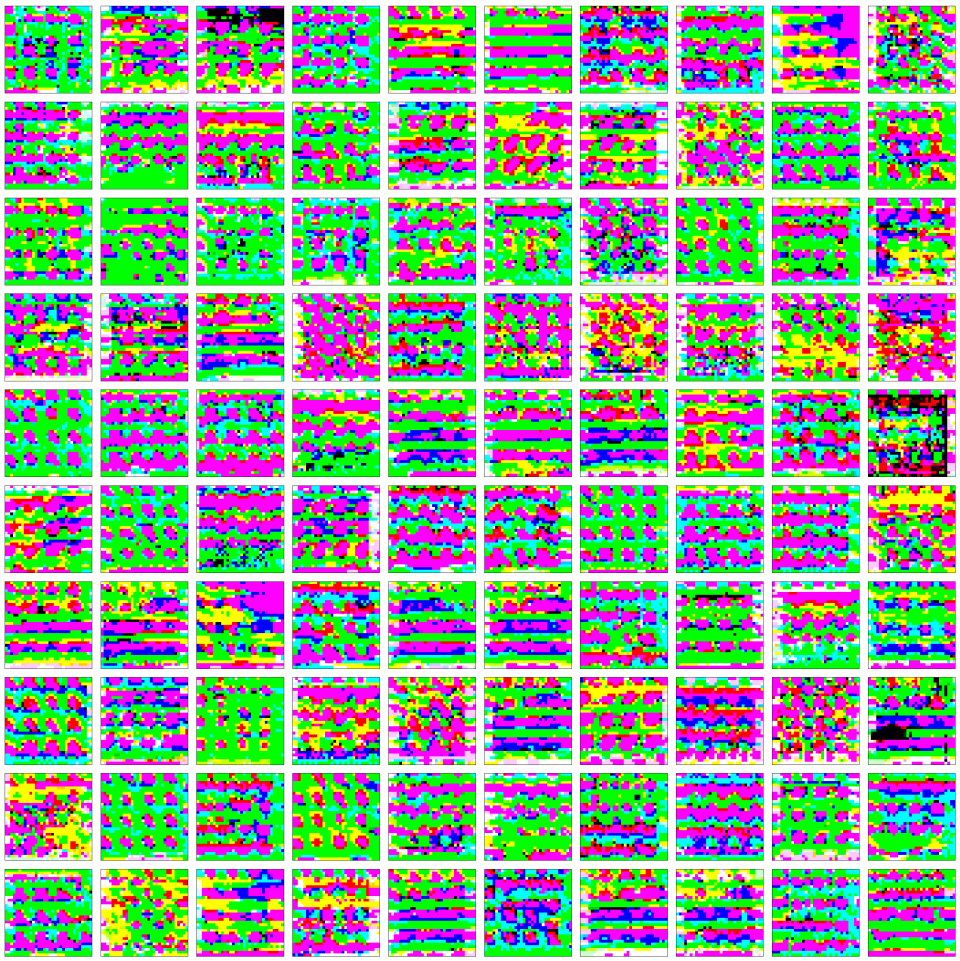


Figure A.2: First 100 sample wise noise masks of the unlearnable CIFAR10 dataset. The error minimizing perturbations are bounded by $\|\delta\|_\infty \leq \frac{8}{255}$

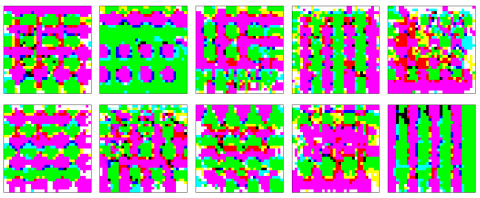


Figure A.3: All ten class wise noise masks of the unlearnable CIFAR10 dataset. The error minimizing perturbations are bounded by $\|\delta\|_\infty \leq \frac{8}{255}$

A.5 Alternative Data Augmentation Techniques

In this section we discuss the resistance of error-minimizing noise against median blur and course dropout. For course dropout we tested two different settings, one where the size of the squares that are dropped out are 11.11% of the image size (course dropout 1), and one where the size of the squares that are dropped out are 6.25% of the total image size (course dropout 2). In both cases only one color channel is dropped out with a probability of 10%, for each block in the image. The course dropout is visualized in Figures A.5c and A.5d of the Appendix.

For median blur we used a kernel size of 3×3. This kernel runs over the image entry by entry, each time replacing the entry with the median of its neighboring entries. The median blur is visualized in Figure A.5e.

We train a ResNet18 model on each of the unlearnable datasets using one of the previously mentioned data augmentation techniques. The test errors of the trained models on a clean test set are depicted in Figure Note that besides course dropout or median blur, no other data augmentation technique is used. Therefore, we have to compare these results to a model trained on clean CIFAR-10 data, where no data augmentation is used. Such a trained model achieves an error of 11.66% on clean test data.

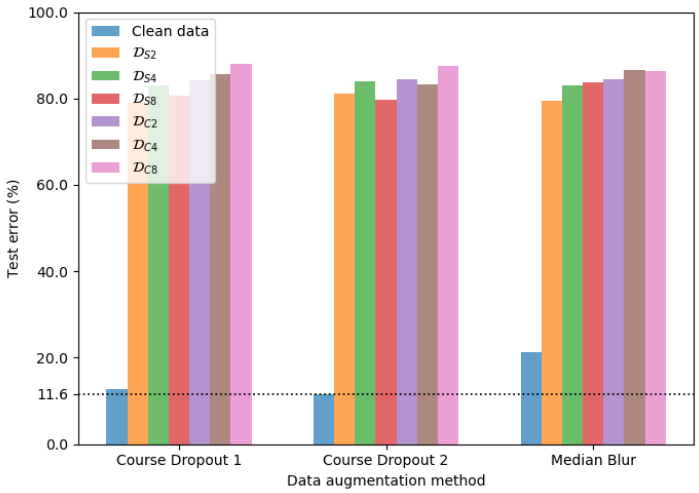
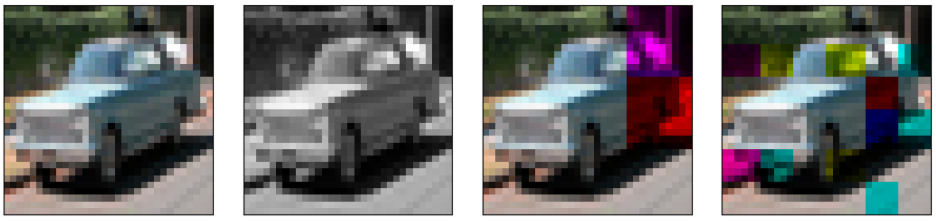


Figure A.4: Test error of a ResNet18 model trained on different unlearnable CIFAR-10 datasets. Samples from the datasets are augmented during training by applying course dropout or median blur. The error of a model trained on clean CIFAR-10 data with no data augmentation is visualized by the dotted line.

All unlearnable datasets are resistant against the three data augmentation techniques. Median blur does improve the performance of the model trained

on $\mathcal{D}_{S_2}$. However, by applying median blur we sacrifice a lot of performance of the model trained on clean data. Therefore, we do not further explore these data augmentation techniques.

A.6 Visualization of Data Augmentation



(a) Natural (b) Grayscale (c) Course Dropout 1 (d) Course Dropout 2

(e) Median blur (f) JPEG $c = 40$ (g) JPEG $c = 60$ (h) JPEG $c = 80$

(i) Noise $\sigma = 0.05$ (j) Noise $\sigma = 0.1$ (k) Noise $\sigma = 0.2$

Figure A.5: Visualization of a different data augmentation techniques used in the experiments