Adversarial examples and loss-task alignment

Experiments on adversarial robustness of models

by Kiarash Farivar

Master's Thesis

July 31, 2022

Contents

1 Introduction

It is a well-known fact that deep learning suffers from susceptibility to adversarial examples [Sze+13]. These examples are optimized to fool the model into misclassifying them only using a small imperceptible amount of noise. They highlight essential shortcomings in the deep learning models and might help us understand them better. Any elements in the deep learning training pipeline can contribute to adversarial susceptibility. These elements are dataset, model architecture, loss function, and optimization method. In this work, I initially studied the effect of different self-supervised *losses* and training methods on the adversarial robustness of deep learning models. Then I changed my focus to the *dataset* and created two toy datasets to eliminate class ambiguity and ensure a robust solution to the problem exists. I trained a model on the toy datasets to observe the effect of using a controlled dataset on model robustness. Finally, I used the Causal3DIdent dataset [Küg+21] to examine model robustness on a more realistic and complicated controlled dataset.

Current works regarding adversarial examples mostly focus on regularizers [AF20], model geometry [Moo+18] or studying intrinsic robustness of models and the features they extract [Ily+19]. However, few works focus on the effect of using different loss functions on the model's adversarial robustness. Self-supervised learning is a training method that encourages a model to extract useful features from samples without using labels. It has made significant improvements recently [Che+20] by demonstrating performance on par with supervised learning methods. Some papers have demonstrated that self-supervised learning can improve different aspects of model robustness. [Hen+19] shows that training a model using the sum of adversarial and self-supervised losses results in improved model robustness, [Zhe+22] uses self-supervised training on datasets with noisy labels to train a supervised model with high accuracy, and [XWM22] proves contrastive learning is robust to labeling noise. In addition, [Ily+19] suggests that the spurious correlation between labels and image features might be a culprit in adversarial susceptibility. Therefore, our initial hypothesis was that training a model using a self-supervised method could reduce the adversarial susceptibility by reducing the model's dependence on labels. Moreover, such a model might extract more robust image features.

To test this hypothesis in section 4 I trained the same encoder network using supervised and different self-supervised training methods on the CIFAR10 classification task; this disproved the hypothesis. To investigate further I used toy and synthetic(Causal3DIdent) datasets in sections 6 and 7 to control the causal factors that affect the dataset creation. The toy dataset consists of small pixel patterns moving on a small canvas. Using it, I could show that in an ideal setting, it is possible to train a 100% robust model with respect to all attacks, using the standard supervised training methods (SGD and crossentropy loss) by just adding the appropriate samples to the dataset. Causal3DIdent is a synthetic dataset created in blender which uses seven objects and different scene properties such as object color, rotation, and lighting angle to render varied images of objects. On Causal3DIdent, I show that even though a model trained using a self-supervised method can retrieve the causal factors of the samples accurately; it can still have an adversarial accuracy of 0%. i.e., Although the model can predict the variables that created an image in the first place, it does not necessarily entail that it is robust.

2 Literature Review [1]

2.1 Adversarial examples

An adversarial example is a sample from the original distribution of the data with some small noise added to it such that the image remains almost identical to the original, but it can fool a model trained on a standard dataset. More formally for a sample $x \in \mathbb{R}^n$ with label y and a classification model $f_\theta(x)$ with parameters θ, we can define the adversarial perturbation $\delta(x) \in \mathbb{R}^n$ as the solution to :

$$\arg\max_{\delta} \ell(f(x+\delta;\theta), y) \tag{1}$$

$$\text{s.t. } \|\delta\| \leq \epsilon \tag{2}$$

Where ℓ is the loss function and the norm of constraint is usually chosen as l_∞ or l_2. Here $x+\delta(x)$ gives the adversarial example. The formulation in 1 is for an **untargeted attack**. The other possibility is a **targeted attack**:

$$\arg\min_{\delta} \ell(f(x+\delta;\theta), y') \tag{3}$$

$$\text{s.t. } y' \neq y \tag{4}$$

$$\|\delta\| \leq \epsilon \tag{5}$$

The attack targets a specific label other than the original (y') and *minimizes* the loss. Note that an untargeted attack can be seen as a targeted attack using the easiest class to fool the model as the target. When solving these optimizations in practice, unlike the formal definition 1, we consider the input as the optimization variable instead of using a separate perturbation δ variable. Note that in the attack optimizations, the model weights θ are kept constant. Different attacks try to solve these optimization problems with different methods. Two of the most famous attacks are FGSM (The fast gradient sign method) and PGD (projected gradient descent) attacks. **FGSM** [GSS14] uses a simple one-step optimization to solve the problem where the perturbation δ is bounded in a l_{inf} ball by ϵ. The adversarial example is defined as:

$$x + \varepsilon \operatorname{sgn}\left(\nabla_x \ell(f(x;\theta), y)\right) \tag{6}$$

PGD [Mad+19] uses multiple steps of projected gradient descent and optionally multiple restarts for x^0 to maximize the loss. This gives it a better chance at finding samples that are hard to reach but comes at the cost of slowing down the process and choosing an appropriate optimization rate α. The PGD steps are defined as:

$$x^{t+1} = \Pi_{x+C}\left(x^t + \alpha \operatorname{sgn}\left(\nabla_x \ell(f(x;\theta), y)\right)\right) \tag{7}$$

Here t is the optimization step, Π is a function that projects the sample back to the closest point inside the neighborhood at each iteration, so the optimization constraints are satisfied, and C

[1] Most of the figures and their caption in this section are taken from the original cited papers.

defines the neighborhood around the sample (i.e., all possible perturbations). For instance, for l_∞ and $\|\delta\| \leq \epsilon$, C will be a hypercube with each side of length ϵ, $x + C$ is the same hypercube centered at x, and Π is the function that truncates each dimension of x, so x would fall inside the hypercube (in case it is not already in the cube).

A more recent ensemble of attacks proposed by [CH20a], called **Autoattack**, tries to mitigate the shortcomings of previous attacks, such as improper tuning of hyper parameters and model gradient obfuscation or masking. They introduce Auto-PGD, which does not require choosing a step size, and an alternative loss called Difference of Logits Ratio(dlr) to resolve some of the problems. These lead to two variants of PGD whose only free parameter is the number of iterations, while everything else is adjusted automatically. They further introduce a set of attacks (including the black-box Square Attack and white-box FAB-attack) to increase attack diversity. In my work, I have only used APGD with the standard cross-entropy loss and 100 iterations for evaluating the models since it already reduces the model robustness for experiments on the CIFAR10 experiments to either near 0 % or random label choice. Note **attacks are similar to counter examples** in the sense that if we find an attack that can reduce model robustness significantly, it proves that the model is not robust. **However**, being robust to a set of attacks does not prove model robustness unless more justification is provided. In the toy dataset section, I only use the attacks to create counter examples and prove model robustness using my assumptions.

2.2 Adversarial training

Adversarial training is the process of training a model that is resistant to an adversarial attack. To my knowledge, training a model resistant to all possible attacks has not been possible. The process of robustly training a network with respect to an attack is identical to the standard training of a neural network, except at each epoch, the adversarial examples replace the training samples that the chosen attack produces. Usually, it is best if the adversarial samples maximize the loss function as much as possible since they represent the worst-case scenario for the network. This method was first suggested by [Mad+19], it is not trivial why it works, and [Mad+19] uses Danskin's theorem to prove the correctness of their method. Robust training is formulated as:

$$\min_\theta \rho(\theta), \quad \text{where} \quad \rho(\theta) = \mathbb{E}_{(x,y)\sim\mathcal{D}}\left[\max_{\delta \in C} \ell(f(x+\delta;\theta), y)\right] \tag{8}$$

Where D is the dataset distribution. We can see that the inner maximization is a set C constrained adversarial example. Note that in the inner optimization, the optimization variable is δ, but in the outer optimization, it is θ. In practice, we alternate between solving the two optimizations and do not optimize them simultaneously. Later, on controlled datasets, I will propose two methods (one mostly theoretical and another more practical) such that training results in a robust model for all attacks for a specific norm and amount of perturbation.

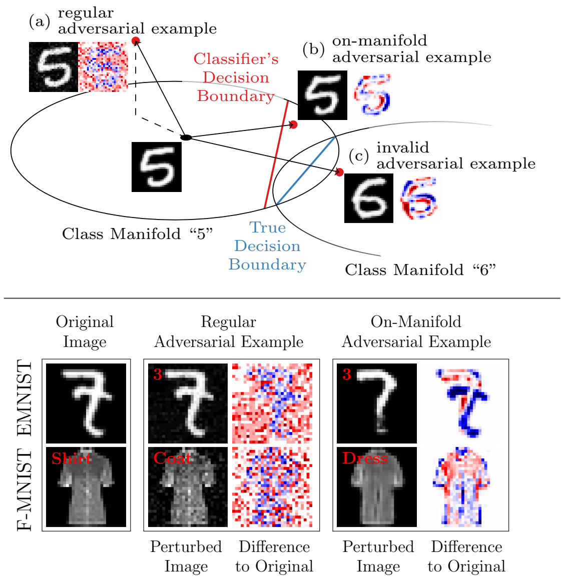

(a) regular adversarial example
Classifier's Decision Boundary
(b) on-manifold adversarial example
(c) invalid adversarial example
Class Manifold "5"
True Decision Boundary
Class Manifold "6"
Original Image
Regular Adversarial Example
On-Manifold Adversarial Example
EMNIST
F-MNIST
3
Shirt
Coat
3
Dress
Perturbed Image
Difference to Original
Perturbed Image
Difference to Original

Figure 1: Adversarial examples, and their (normalized) difference to the original image, in the context of the underlying manifold, eg. class manifolds "5" and "6" on EMNIST, allow to study their relation to generalization. Regular adversarial examples are not constrained to the manifold, cf. (a), and often result in (seemingly) random noise patterns; in fact, we show that they leave the manifold. However, adversarial examples on the manifold can be found as well, cf. (b), resulting in meaningful manipulations of the image content; however, care needs to be taken that the actual, true label w.r.t the manifold does not change, cf. (c).

2.3 On-manifold vs Off-manifold Adversarial Robustness

It can be useful to make a distinction between two main types of adversarial examples: On-manifold and Off-manifold as suggested by [SHS19]. On-manifold samples are essentially samples taken from the data distribution that also generated the training set and off-manifold samples are samples that have probability 0 of occurring (according to the data distribution). The paper argues that the standard method of creating adversarial examples (2.1 the paper calls them "regular adversarial examples") creates samples that are out of distribution (off-manifold). They show this in the toy FONTS dataset were they define the data manifold themselves, sample from it, then applying the attacks and finally calculate the distance between the attack image and the closest point on the data manifold. Most adversarial images are more than 0.5 away from the manifold in norm l_2. They also show this on more realistic datasets (EMNIST, F-MNIST and CelebA) by approximating the data manifold using VAE-GANs. They also show that regular adversarial examples leave the manifold in an almost orthogonal direction.

They define **on-manifold adversarial examples** $\tilde{x}$ for an image x with label y as on-manifold samples that according to the conditional data distribution $p(y|x)$ belong to the same class as x but are misclassified by the model $f(x)$.

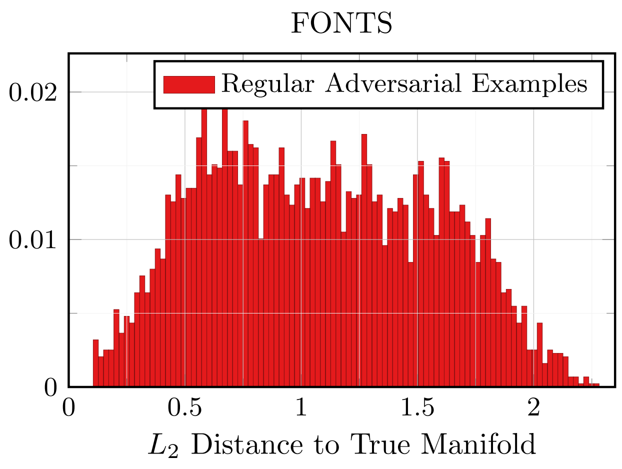


Figure 2

$$\forall y' \neq y \quad p(y|\tilde{x}) > p(y'|\tilde{x}) \tag{9}$$
$$f(\tilde{x}) \neq y \tag{10}$$

Obviously training on-manifold adversarial examples improves generalization and model accuracy since these samples come from the data distribution. In practice instead of approximating the manifold using generative models, we can exploit known invariances of the data. Then, adversarial training can be applied to these invariances, assuming that they are part of the true manifold (e.g. translation, shear, scaling and rotation for images).

They further argue that generalization (clean accuracy) and Robust accuracy are not conflicting goals. They explain this by the fact that regular adversarial examples leave the manifold so as a result, the network has to learn (seemingly) random, but adversarial, noise patterns in addition to the actual task at hand; rendering the learning problem harder.

2.4 Self-supervised learning (SSL)

Self-supervised learning is the process of training a model(called encoder or backbone and shown as $f(.)$) using a pretext task, without using any human labeled samples. The model maps samples to a vector of useful features and their usefulness is assessed by their performance in downstream tasks (e.g. classification). To adapt the encoder for use in a downstream task, while keeping the encoder fixed, we add a simple model (e.g. linear) after the encoder output and train only the simple model on a much smaller labeled dataset (see 12). One possible pretext task is rotation, [GSK18] randomly rotates images and then the encoder network predicts the image rotation. They mention since the network needs to make use of important image features for this task the representation learned by it will be useful in downstream tasks. Therefore, this network can subsequently be used to perform classification.

The more recent methods use image augmentation and the InfoNce loss [2] to train an encoder network. They are called **contrastive learning** methods, the most famous of which is simCLR [Che+20]. In simclr to each standard sample x two random augmentations are applied (for images this usually includes random crop and color distortion) and two versions of the sample are created $\tilde{x_i}$ and $\tilde{x_j}$. The goal of the network according to the loss is to find representations such that the two versions have encodings that are as close as possible using cosine similarity $(\mathrm{sim}(\boldsymbol{u}, \boldsymbol{v}) = \boldsymbol{u}^\top \boldsymbol{v}/\|\boldsymbol{u}\|\|\boldsymbol{v}\|)$ and dissimilar to other images. A visual representation of the framework can be seen in 3. They further discovered that adding a projection head $g(.)$, only during training, can make the encodings more useful since the features are not strictly fitted for minimizing their loss. The loss function for a single standard image in a batch is defined as:

$$\ell_{i,j} = -\log \frac{\exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{k \neq i} \exp(\mathrm{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)}\,, \tag{11}$$

The total loss is the sum of the individual loss over all samples [3]. In InfoNCE loss notation the two versions ("views") of the standard image correspond to the positive pairs represented by x and x^+ and the other images (in the denominator other than z_j) constitute the negative samples x^-. The remarkable fact is that although images from the same class (defined in an unknown and unseen downstream task) can be considered by the loss as positive and negative pairs the model is still able to extract features such that they are useful in the downstream tasks [4].

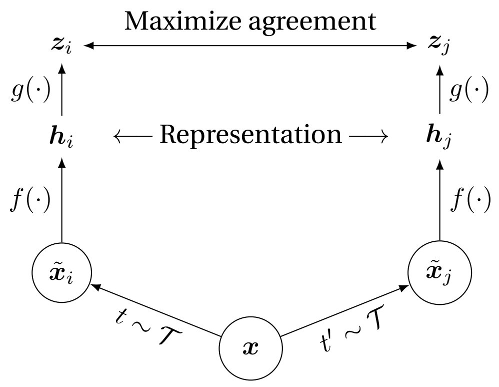


Figure 3: SimCLR, a simple framework for contrastive learning of visual representations. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement using a contrastive loss. After training is completed, we throw away the projection head $g(\cdot)$ and use encoder $f(\cdot)$ and representation $\boldsymbol{h}$ for downstream tasks.

[2]see [Aro+19] definition 2.3 for definition

[3]We we add both $\ell(i,j)$ and $\ell(j,i)$ to make use of all samples and to be symmetric also the sum in denominator is until $2N$ since each sample has 2 augmented views.

[4]The literature is mostly obsessed with *standard* classification as the downstream task but an interesting question might be to come up with tasks such that the model doesn't have a good performance in them. One such task is robust classification !

Barlow twins [Zbo+21] is another surprising SSL method that doesn't rely on contrastive learning. Instead the loss is created using the cross correlation matrix. The setup is same as before except for the loss. A visual representation of the method can be seen in 4 the loss is defined as :

$$\mathcal{L}_{\mathcal{BT}} \triangleq \underbrace{\sum_i (1-\mathcal{C}_{ii})^2}_{\text{invariance term}} + \lambda \underbrace{\sum_i \sum_{j \neq i} {\mathcal{C}_{ij}}^2}_{\text{redundancy reduction term}} \tag{12}$$

where λ is a positive constant trading off the importance of the first and second terms of the loss, and where $\mathcal{C}$ is the cross-correlation matrix computed between the outputs of the two identical networks along the batch dimension:

$$\mathcal{C}_{ij} \triangleq \frac{\sum_b z^A_{b,i} z^B_{b,j}}{\sqrt{\sum_b \left(z^A_{b,i}\right)^2} \sqrt{\sum_b \left(z^B_{b,j}\right)^2}} \tag{13}$$

where b indexes batch samples and i, j index the vector dimension of the networks' outputs. $\mathcal{C}$ is a square matrix with size the dimensionality of the network's output, and with values comprised between -1 (i.e. perfect anti-correlation) and 1 (i.e. perfect correlation).

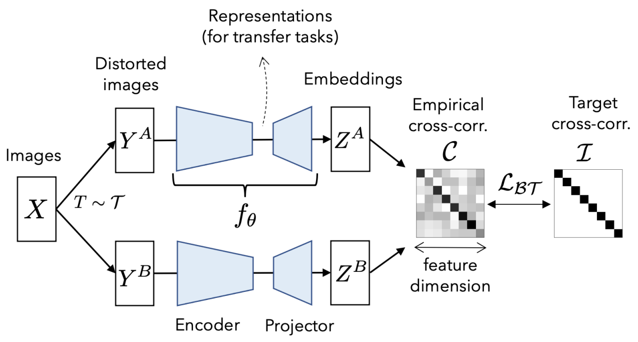


Figure 4: Barlow Twins' objective function measures the cross-correlation matrix between the embeddings of two identical networks fed with distorted versions of a batch of samples, and tries to make this matrix close to the identity. This causes the embedding vectors of distorted versions of a sample to be similar, while minimizing the redundancy between the components of these vectors. Barlow Twins is competitive with state-of-the-art methods for self-supervised learning while being conceptually simpler, naturally avoiding trivial constant (i.e. collapsed) embeddings, and being robust to the training batch size.

Simsiam belongs to a category of SSL models where similar to contrastive learning they use positive pairs but don't use negative pairs. To prevent the model from creating trivial solutions such as mapping every sample to a single vector (known as collapsing) they use a stop-gradient operation. The framework is similar to simclr except the addition of the stop-gradient and a prediction head which makes the branches asymmetric. Note that in their notation $f(.)$ includes

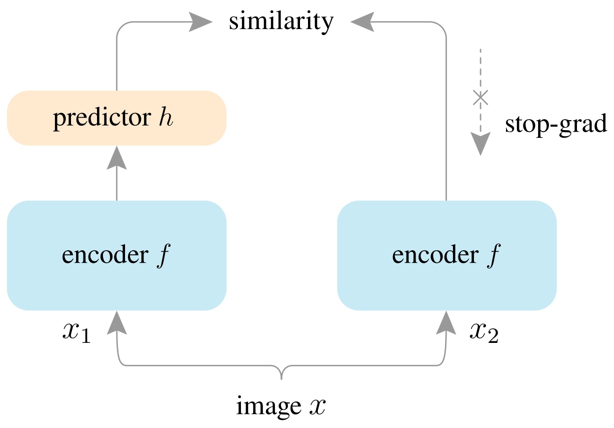


Figure 5: **SimSiam architecture**. Two augmented views of one image are processed by the same encoder network f (a backbone plus a projection MLP). Then a prediction MLP h is applied on one side, and a stop-gradient operation is applied on the other side. The model maximizes the similarity between both sides. It uses neither negative pairs nor a momentum encoder.

both the encoder and the projection head. Their pipeline can be seen in 5. Their architecture takes as input two randomly augmented views x_1 and x_2 from an image x. The two views are processed by an encoder network f consisting of a backbone (e.g. ResNet) and a projection MLP head. The encoder f shares weights between the two views. A prediction MLP head, denoted as h, transforms the output of one view and matches it to the other view. Denoting the two output vectors as $p_1 \triangleq h(f(x_1))$ and $z_2 \triangleq f(x_2)$, we minimize their negative cosine similarity:

$$D(p_1, z_2) = -\frac{p_1}{\|p_1\|_2} \cdot \frac{z_2}{\|z_2\|_2}, \tag{14}$$

where $\|\cdot\|_2$ is ℓ_2-norm. This is equivalent to the mean squared error of ℓ_2-normalized vectors up to a scale of 2. They define their symmetrized loss as:

$$\mathcal{L} = \frac{1}{2} D(p_1, z_2) + \frac{1}{2} D(p_2, z_1). \tag{15}$$

This is defined for each image, and the total loss is averaged over all images. Its minimum possible value is -1. An important component for their method to work is a stop-gradient (`stopgrad`) operation. They implement it by modifying (14) as:

$$D(p_1, \texttt{stopgrad}(z_2)). \tag{16}$$

This means that z_2 is treated as a constant in this term. Similarly, the form in (15) is implemented as:

$$\mathcal{L} = \frac{1}{2} D(p_1, \texttt{stopgrad}(z_2)) + \frac{1}{2} D(p_2, \texttt{stopgrad}(z_1)). \tag{17}$$

Here the encoder on x_2 receives no gradient from z_2 in the first term, but it receives gradients from p_2 in the second term (and vice versa for x_1).

An important **note** regarding the SSL methods: there are extra layers added on top of the encoder during SSL training, such as the projection layer and the prediction layers (only in simsiam). These are only used during SSL training and then thrown away. In later sections to fine-tune the models for a classification task I will add a layer on top of the encoder called the readout layer this has nothing to do with the extra SSL training layers whatsoever.

Other papers have attempted to prove why the encoding created by contrastive learning will be useful on an unseen downstream task when only a linear readout layer is fine-tuned for the task. [Aro+19] proves that with a high probability (similar to PAC learning) it is possible to upper bound the loss of the learned supervised classifier if the encoder is trained using contrastive learning. Unrealistically they assume that x and x^+ are independent given the label, this is obviously not the case for methods like simclr where x and x^+ are augmentations of the same image. However they realistically assume that we might sample negative data points x^- from the same class as x. The simple version of the upper bound (section 4.1 in their paper) is dependent on: the probability that two classes sampled independently from the class distribution are the same, the the unsupervised contrastive loss of the encoder on the whole data distribution and the Radmacher complexity of linear classification task on the encoded samples dataset. [Hao+21] uses augmentation graphs and spectral graph theory to do the proof without using the fact that x and x^+ need to be independent given the label. Augmentation graphs represent augmented images as vertices and two images are connected with an edge if they are augmentations of the same natural image. They introduce a loss such that when it is minimized (instead of the contrastive loss) gives an encoder that can be used to represent each sample. Each representation vector is placed in a row of a matrix, call this the representation matrix. They show that the population representation matrix can be decomposed using the eigenvectors of the normalized adjacency matrix of the augmentation graph. Their main result shows that when the representation dimension exceeds the maximum number of disconnected sub-graphs, linear classification with learned representations is guaranteed to have a small error.

2.5 Shortcut Solutions and Feature Suppression in SSL

There has been investigations of supervised models using shortcut solutions such as [Gei+18] where they show supervised CNN models are biased towards recognizing textures rather than shapes which is in contrast to how humans recognize objects in images. But for SSL models [CLL20] seems to be the first paper to investigate this effect. In their section 4 they argue that contrastive learning requires good design of data augmentation to work well (other papers also prove that good augmentations are required for disentanglement see 2.8) . For instance, if we don't use color distortion in the augmentation step of contrastive learning, the easier to detect feature of color in images can suppress the ability of the model from detecting features that are important for a downstream object classification task thus reducing the quality of representations. The problem is that: *there may be scenarios where the known augmentations cannot fully address the feature suppression effect, and it can thus limit the potential of contrastive learning*. In the most interesting experiment of their section they create the **DigitOnImageNet** dataset by overlaying MNIST digits on top of Imagenet samples. For each Imagenet image they assign a unique MNIST digit and replicate it in nine fixed locations before the standard SimCLR

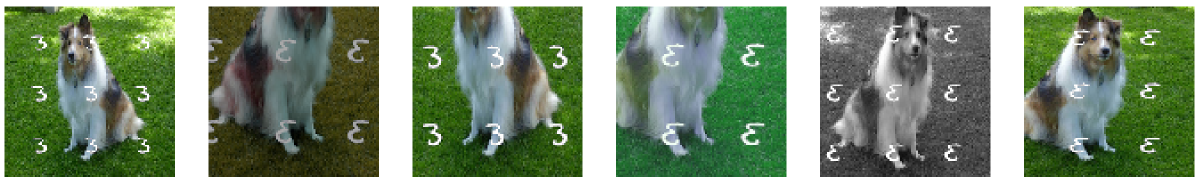
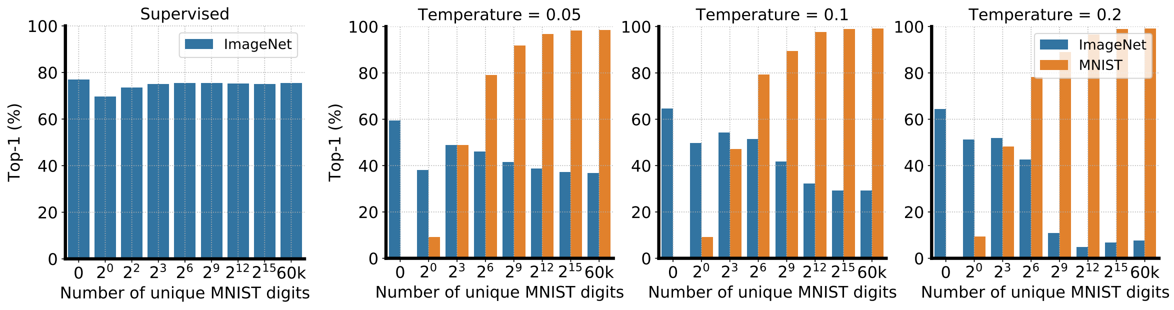


Figure 6: A sample from DigitOnImageNet



(a) Supervised learning (b) Unsupervised contrastive learning

Figure 7: (a) Supervised learning accuracy on ImageNet classification. (b) Linear evaluation of learned features for both MNIST classification and ImageNet classification on the DigitOnImageNet dataset. Batch size of 1024 and 2-layer projection head is used. Different batch sizes and projection head layers have negligible influence on the trade-off between ImageNet vs MNIST accuracy.

augmentations are applied to create augmented views (see example 6). Therefore, the features of the two datasets are competing features. The best outcome is that the model learns both set of features so it can perform well on the classification tasks of both datasets.

They change the number of unique digits used in the training set, while all MNIST digits are used in the validation/test set. In supervised learning the number of unique digits has little to no impact on the classification accuracy however, in simclr, using different temperatures, as we increase the number of unique images the accuracy of MNIST classification increases while that of Imagenet decreases dramatically (see 7). This effect is worse in higher temperatures. The trade-off between digit recognition and object recognition abilities shows that simple features suppress the learning of difficult features, when both are shared between two augmented views. This has important implications for the robustness of contrastive and more generally SSL methods. Since, if a simple non-adversarial modification, such as adding digits on top of images, can effect encoding quality to this degree then there is little hope for innate adversarial robustness of such a training method.

2.6 Robustness of contrastive learning

There has been recent papers showing that contrastive learning can be robust to different types of noise. [Hen+19] mentions how SSL loss can improve robustness w.r.t common corruptions and adversaries. The improvement in robustness on adversarial examples was shown on CIFAR10, the big **caveat** is that they use the adversarial training loss [Mad+19] (as in 18) and add the SSL loss to the adversarial loss to further improve robustness. Their SSL loss is based on predicting image rotations as in [GSK18]. Compared to only using the adversarial loss adding the SSL loss improves model robustness from 1% for $\epsilon = 4/255$ to 10% for $\epsilon = 10/255$ (their figure 2). As I will show later using the SSL loss alone yields no robustness on CIFAR10. They further demonstrate that using their loss the model acquires robustness with respect to common corruptions such as noise, blur, zoom blur and JPEG compression. Obviously these corruptions are not targeted attacks so it is much simpler for the model to be robust to them.

$$\mathcal{L}(x, y; \theta) = \mathcal{L}_{\text{CE}}(y, p(y \mid \text{PGD}(x)); \theta) + \lambda \mathcal{L}_{\text{SS}}(\text{PGD}(x); \theta) \tag{18}$$

Another line of work analyses using SSL to mitigate the effect of labeling noise on model performance. [Zhe+22] considers the obstacle of warm-up stage in "learning with noisy labels (LNL)" training stage. The problem is that current methods adjust the warm-up lengths based on the observed robustness of the model under different noise levels, and determining the number of optimal warm-up epochs is non-trivial. Their method Contrast to Divide(C2D) uses both ELR and simclr to achieve consistently high accuracy under different noise rates and types, with markedly improved performance under very-high noise conditions. The advantage of using SSL compared to supervised training (and other supervised variants designed for LNL) in an LNL setting is that the sample's latent representation is not affected by labeling noise at all. Early-Learning Regularization (ELR) [Liu+20] is a supervised LNL method that gradually increases the weight of a regularization term in the loss to prevent the model from memorizing noisy samples with wrong labels. [XWM22] uses augmentation graphs to prove contrastive learning is robust to labeling noise. For instance introducing 80% symmetrical label noise on CIFAR10 dataset while using their method can give us a model that has an accuracy above 70%.

2.7 Robust Self-Supervised Training

Later I will show SSL models are not inherently robust for all perturbations specifically against adversarial examples. There are two papers corresponding to two different perspectives on SSL training for feature robustness: [Rob+21] discusses SSL robustness in context of in-distribution (on-manifold) robustness and [Jia+20] where they adversarially train the the contrastive model and is similar to supervised adversarial training.

2.7.1 Can contrastive learning avoid shortcut solutions?

[Rob+21] asks the question "can the contrastive instance discrimination task itself be modified to avoid learning shortcut solutions?"

Just like in supervised learning, unsupervised learning can also suffer from feature suppression. They can also suffer from simplicity bias. Both supervised and contrastive learning suffer from biases induced by the choice of optimizer and architecture. But they study how modifying the loss can affect the result of unsupervised learning.

In sections 2.1 and 2.2 and 2.3 they mathematically show that minimizing the InfoNCE loss can lead to feature suppression (proposition 1). They also show if a feature is constant (or similar) in a batch (both among the positive and negative samples) then it is not used by the model to discriminate that batch (proposition 2).

They study two existing methods that try to resolve the problem: Changing the temperature τ in the InfoNCE loss and Hard negative sampling. Both methods cause a trade off between the easy(e.g. color) and hard (e.g. shape) features, since the features that are held constant are suppressed.

In section 3 they suggest that we should adversarially perturb the output encoding that the model produces (logits) to adjust which feature a model pays attention to. They call it Implicit Feature Modification (IFM). In this way they avoid a trade off since in a sense they adaptively change which feature is removed or held constant. It is possible to calculate an exact solution to the gradient and explicitly solve the optimization. At the end this amounts to simply perturbing the logits (inner products); reduce the positive logit by ϵ^+/τ and increase negative logits by ϵ_i/τ. IFM re-weights each negative sample by a factor $e^{\epsilon_i/\tau}$ and positive samples by $e^{\epsilon^+/\tau}$.

According to their notation, given batch $x, x^+, \{x_i^-\}_{i=1}^m$ we write $v = f(x)$, $v^+ = f(x^+)$, and $v_i^- = f(x_i^-)$ to denote the corresponding embeddings. The point-wise InfoNCE loss is,

$$\ell(v, v^+, \{v_i^-\}_{i=1}^m) = -\log \frac{e^{v^\top v^+/\tau}}{e^{v^\top v^+/\tau} + \sum_{i=1}^m e^{v^\top v_i^-/\tau}}.$$

Definition:(Implicit feature modification) Given budget $\boldsymbol{\varepsilon} \in \mathbb{R}_+^m$, and encoder $f : \mathcal{X} \to \mathbb{S}^d$, an adversary removes features from f that discriminates batch $x, x^+, \{x_i^-\}_{i=1}^m$ by maximizing the point-wise InfoNCE loss, $\ell_{\boldsymbol{\varepsilon}}(v, v^+, \{v_i^-\}_{i=1}^m) = \max_{\delta^+ \in \mathcal{B}_{\varepsilon^+}, \{\delta_i^- \in \mathcal{B}_{\varepsilon_i}\}_{i=1}^m} \ell(v, v^+ + \delta^+, \{v_i^- + \delta_i^-\}_{i=1}^m)$.

Here $\mathcal{B}_\varepsilon$ denotes the ℓ_2-ball of radius ε. Finally the modification to the loss is given as:

$$\ell_{\boldsymbol{\varepsilon}}(v, v^+, \{v_i^-\}_{i=1}^m) = -\log \frac{e^{(v^\top v^+ - \varepsilon^+)/\tau}}{e^{(v^\top v^+ - \varepsilon^+)/\tau} + \sum_{i=1}^m e^{(v^\top v_i^- + \varepsilon_i)/\tau}}. \tag{19}$$

Their improvements on real datasets seems insignificant in many cases except when they use big images (Imagenet) and have lots of classes (figure 5 CIFAR100 and 7 tinyImageNet with 200 classes). This is in contrast to the toy dataset they use (Trifeature dataset) where they get significant improvements (figure 6). It is also worth mentioning that their method doesn't significantly improve feature suppression in the STL-digits dataset with 10 classes (Figure 14 appendix). STL-digits is an important dataset designed specifically to analyze feature suppression in SSL created by [CLL21a].

Note that their method is not effective for adversarial robustness since an epsilon perturbation in the input can create a neighborhood in the feature space that is not an l_p norm ball. see 8.

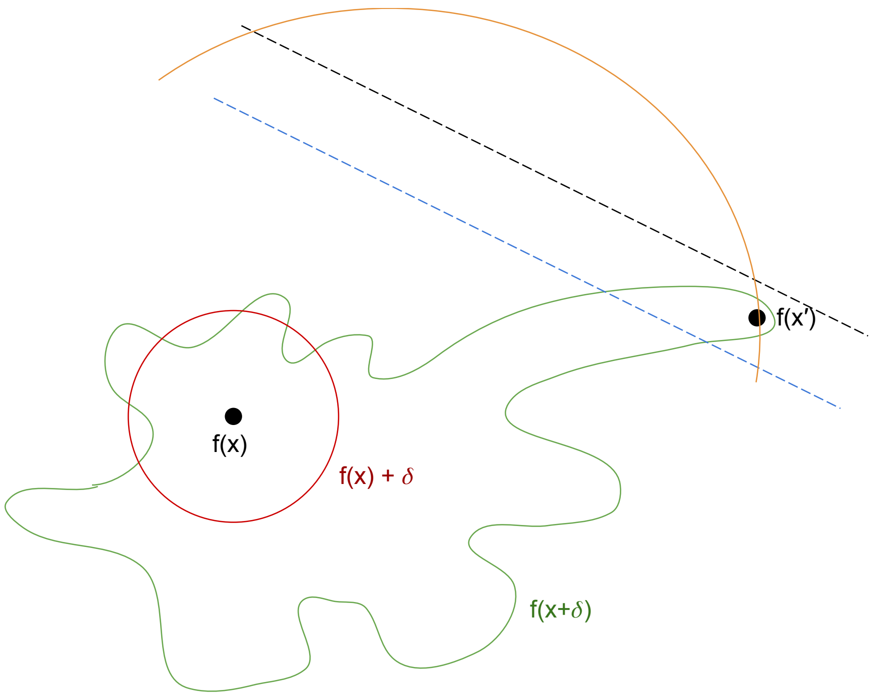


Figure 8: An example of the feature space. The standard image x is mapped by the model in the feature space to f(x). The ball considered by IFM ($f(x) + \delta$) is shown in red (assuming l_2 norm). The neighborhood covered by the adversarial perturbation is shown in green ($f(x + \delta)$). The black dashed line is the correct boundary and the blue dashed line is the boundary learned by the model. $f(x')$ is the worst case adversarial example found using an attack. The orange curve is the radius of the ball if IFM wanted to include the adversarial sample $f(x')$. This would obviously force the model to incorrectly assign many samples to the same class as f(x).

2.7.2 Robust Pre-Training by Adversarial Contrastive Learning

[Jia+20] presents 3 methods to adversarially train a contrastive learning method (e.g. Simclr). In figure 9 we can see the different pipelines. As per their notation for Simclr the two randomly augmented versions of the image x is represented by ($\tilde{\boldsymbol{x}}_i$, $\tilde{\boldsymbol{x}}_j$).

For **Adversarial-to-Adversarial (A2A)** training they generate ($\boldsymbol{\delta}_i$, $\boldsymbol{\delta}_j$) as adversarial perturbations corresponding to the augmented samples ($\tilde{\boldsymbol{x}}_i$, $\tilde{\boldsymbol{x}}_j$), using the PGD attack algorithm , respectively. The loss maximized to calculate this perturbation is the contrastive loss. In other words, they generate two views that are adversarially enforced to be dissimilar, on which they learn to enforce consistency. In this way, the contrastive learning is now performed on ($\tilde{\boldsymbol{x}}_i + \boldsymbol{\delta}_i$, $\tilde{\boldsymbol{x}}_j + \boldsymbol{\delta}_j$) instead of ($\tilde{\boldsymbol{x}}_i$, $\tilde{\boldsymbol{x}}_j$).

For **Adversarial-to-Standard (A2S)** training they feed one of the branches an adversarial example $\tilde{\boldsymbol{x}}_i + \boldsymbol{\delta}_i$ while keeping the other branch standard $\tilde{\boldsymbol{x}}_j$. Since the statistics of the adversarial images are different than that of the standard images they use separate BN parameters for each branch but keep the convolutional parameters the same. Finally they use the encoder from the adversarial branch to get the best robustness.

Dual Stream (DS) combines the two methods to create two separate simclr training instances one standard and the other advresarial. The 4 bracnhes share all convolutional parameters but the BN is separate for the adversarial instance. Finally they optimize the average of the standard loss (from the standard instance) and the adversarial loss.

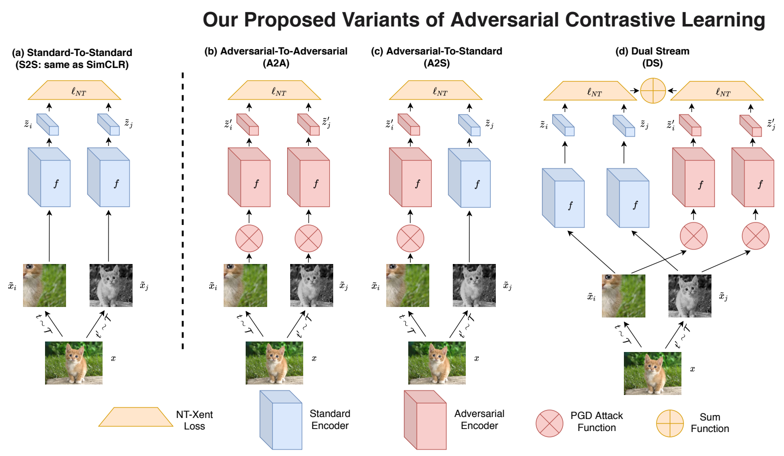

Our Proposed Variants of Adversarial Contrastive Learning
(a) Standard-To-Standard (S2S: same as SimCLR)
(b) Adversarial-To-Adversarial (A2A)
(c) Adversarial-To-Standard (A2S)
(d) Dual Stream (DS)
ℓ_{NT}
f
x
NT-Xent Loss
Standard Encoder
Adversarial Encoder
PGD Attack Function
Sum Function

Figure 9: (a) The original SimCLR framework, a.k.a., standard to standard (no adversarial attack involved); (b) - (d) three proposed variants of the adversarial contrastive learning framework: A2A, A2S, and DS (their best solution). Note that, whenever more than one encoder branches co-exist in one framework, they by default share all weights, except that adversarial and standard encoders will use independent BN parameters.

They show that their method yields an improvement of [2.14%, 2.99%] on CIFAR-10, and [2.14%, 3.58%] on CIFAR-100, for Standard test accuracy and Robust accuracy respectively, compared to the previous methods. Considering that the robust accuracies are around 50% and the huge computational cost of their method I am skeptical regarding the significance of these results.

2.8 SSL and Latent Variable Disentanglement

Assume we have a dataset such that each sample is created according to a causal graph 10. Latent Variable Disentanglement can be defined as identifying the causal factors that created the sample in the first place. [Küg+21] separates these factors into content which is shared or invariant across augmented views and style that may change. They show in their theorem 4.4 that it is possible to have a model that retrieves the content variable of samples using a special loss function which uses differential entropy as a regularizer. They mention that contrastive SSL with negative samples using InfoNCE loss as an objective can asymptotically be understood as alignment with entropy regularization. Therefore, they prove that subject to their assumptions, contrastive learning with InfoNCE asymptotically isolates content, i.e., the part of the representation that is always left invariant by augmentation. In my work I will show that a model that is able to retrieve causal factors is not necessarily robust. They also introduce the Causal3DIdent dataset which is a causal dataset that I use to evaluate model robustness.

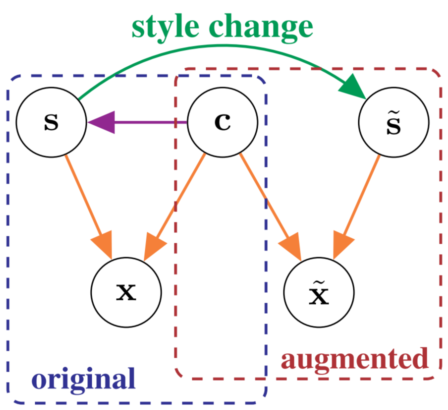


Figure 10: The latent variable z is partitioned into content c and style s, and they allow for statistical and causal dependence of style on content(purple arrow). It is assumed that only style changes (green arrow) between the original view x and the augmented view $\tilde{x}$, i.e., they are obtained by applying the same deterministic function (orange arrows) f to $z = (c, s)$ and $\tilde{z} = (c, \tilde{s})$.

3 Hypothesis

Training a neural network consists of 3 main components: The model architecture, The dataset, and the optimization method. So the influence of all these components can also contribute to the phenomenon of adversarial examples. For instance, a convolutional network has a specific architecture that could affect the smoothness of the boundary, or separate classes in the dataset could be overlapping which makes the samples in the overlap region ambiguous and gives a theoretical upper bound for the model robustness. My initial focus in section 4 was on how defining different losses (supervised vs. SSL) could affect model robustness. We hypothesized that an SSL model would be more robust than a supervised model. Later I created some toy datasets to examine the contribution of the dataset on robustness. In section 6 I hypothesized that it is possible to train a robust model using a standard architecture and training method given an idealistic dataset. To investigate the failure of SSL models in being robust on CIFAR10, in section 7 I used a dataset with an explicit causal graph hypothesizing that if the model can accurately retrieve the causal factors of samples, it will be more robust.

Here I will provide evidence for why SSL training might be more robust than supervised training. Many papers have focused on regularizing the supervised loss or studying the model's boundary geometry or features in the dataset to analyze model robustness. [AF20] introduces a new regularization method called GradAlign to prevent catastrophic overfitting when adversarially training a model using FGSM. They do this by maximizing the gradient alignment inside the perturbation set. [Moo+18] investigates how the curvature of the model's boundary is related to robustness (i.e., adversarially trained models have lower curvatures than standard supervised models). They then introduce a new regularization method, Curvature Regularization (CURE), that encourages the model to learn a low curvature boundary, and they show it achieves robustness levels comparable to that of adversarial training. Other papers study intrinsic robustness and the features in the data which contribute to robustness or adversarial susceptibility [Ily+19]. But what about the effect of the **loss function** on the robustness of a model?

Our initial intuition was that the robustness problem encountered in a supervised setting is due to using labeled data. As suggested by [Ily+19] and [Tsi+19], the spurious correlation between classes and non-robust image features could be strongly connected to adversarial susceptibility, which exists due to the existence of a label for each image. [Jac+18] shows that, in a supervised classification task, it is possible to get the same logit outputs from a model for images from entirely different classes (e.g., frog vs. monkey). They do it by perturbing images such that their human-recognizable features are fixed while the non-class related features of images are made as close as possible. They call these perturbed images Invariance-based Adversarial Examples, and the shortcoming of the model is called excessive invariance. They hypothesize that a significant reason for this excessive invariance can be understood via the information-theoretic viewpoint of cross-entropy, which maximizes a bound on the mutual information between *labels* and representation, giving no incentive for the model to explain all class-dependent aspects of the input. Furthermore, there could also exist ambiguous and unnatural separation of classes in the dataset due to labeling. One reason might be that most classification tasks use hard labeling (e.g., a sample is either 100% dog or 100% cat) instead of assigning soft labels, specially for ambiguous samples (e.g., a single image could be classified as 80% dog and 20% cat). Another reason could be that there always exists the possibility of mislabeling samples due to human error in a human-labeled dataset (a.k.a labeling noise). All of these factors, which affect the training via the labels, can lead to the boundary learned by the model to be unnecessarily distorted and less smooth, which in turn could contribute to more adversarial vulnerability (fig 11). So it is natural to ask:

Is a self-supervised model more robust than a supervised model ? and Do SSL models retain more robust features than supervised models ?

My initial goal in section 4 was to:

1. Compare the classification accuracy of different loss functions and training methods on the CIFAR10 dataset using different attacks.
2. Compare the norm change of representations learned by self-supervised losses and the supervised loss on CIFAR10.

4 Self-supervised learning, not robust on CIFAR10

Our initial hypothesis was that training a self-supervised model could reduce adversarial susceptibility by removing the spurious correlation between images and labels. Furthermore, an SSL model would retain more useful features due to the lack of labels. I trained supervised and different SSL models on the CIFAR10 classification task to test this hypothesis. The unsupervised training of each SSL model is done according to the specific design of the method. To fine-tune the SSL models for classification, I trained a final linear layer (added after the encoder) while keeping the encoder fixed as shown in 12. The details are as follows.

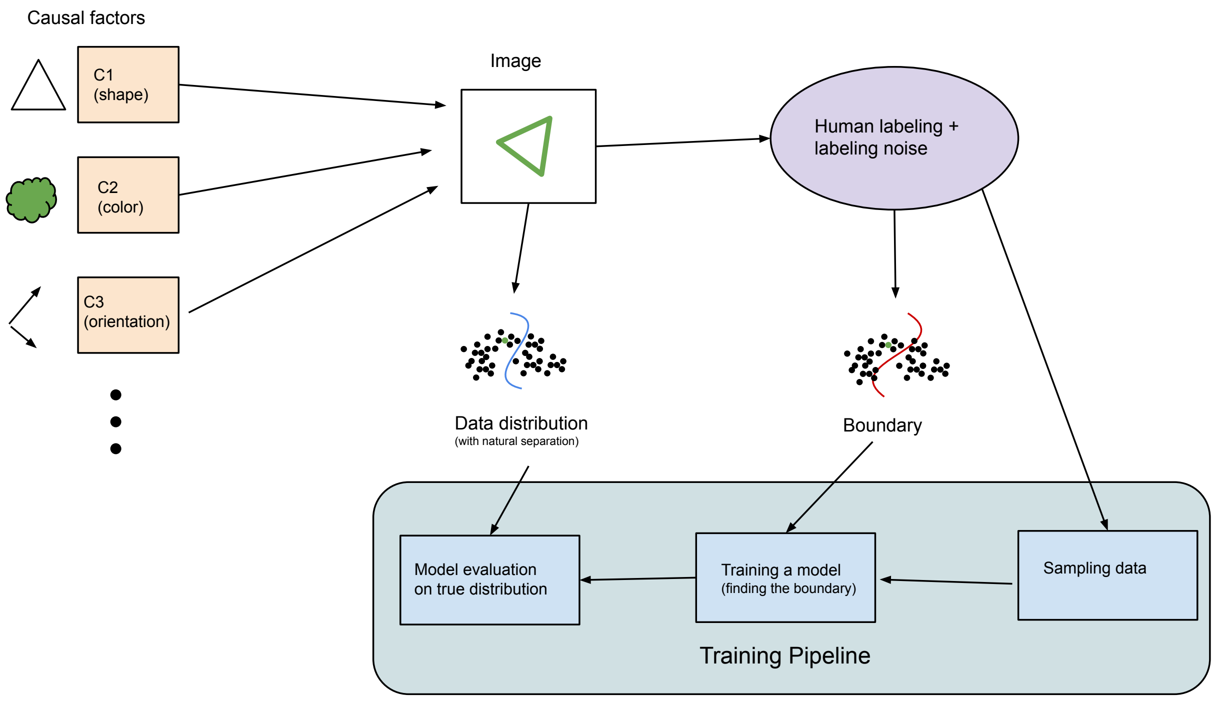

Causal factors
C1 (shape)
C2 (color)
C3 (orientation)
Image
Human labeling + labeling noise
Data distribution
(with natural separation)
Boundary
Model evaluation on true distribution
Training a model (finding the boundary)
Sampling data
Training Pipeline

Figure 11: Many causal factors can contribute to creating a dataset; these are shown with brown boxes on the top left. To create classification tasks, humans usually label samples, so they introduce their own biases to the dataset via the labels. The blue boundary shows natural data separation; the red boundary is distorted due to labeling noise which could contribute to adversarial susceptibility. Finally, we sample the dataset to train the model according to the labels but evaluate on unseen samples from the real distribution. Could this contribute to model's adversarial susceptibility ?

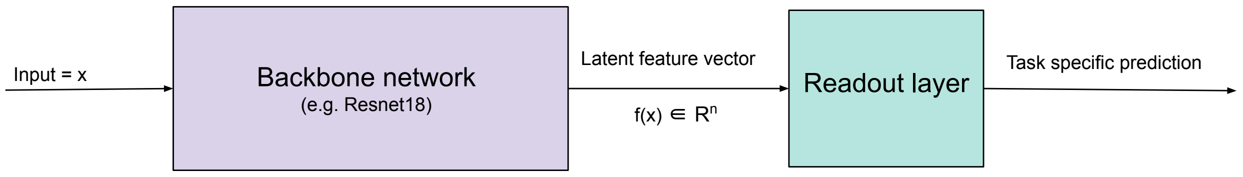


Figure 12: SSL model parts.

4.1 Models and training

All the models were modified as needed for the CIFAR10 classification task (refer to code) but their implementation are faithful to the original design in the papers. The models are:

1. bralow twins.
2. simCLR (implementation by lightning bolts [FC20])
3. simsiam (implementation by lightning bolts [FC20])
4. supervised (implementation by [Pha21]).

For the **backend** network, all methods use Resnet18 (18-layer convolutional network with skip connections), as implemented in the lightning bolts library, which has the output dimension 512. All backends were trained according to the implementation of the repositories, which they came from with necessary modifications.

For **SimCLR** [Che+20] the projection head has 1 hidden layer with 2048 units and output size 128. All parameters are the same as the original paper except for the number of layers in the projection head. The paper uses two layers for it when training on Imagenet. However, since I train on CIFAR10, which has a much smaller image resolution, and to reduce training time, I used one layer[5]. Similar to the paper, I used LARS for optimization.

Simsiam [CH20b] has the same parameters for projection head as simclr except that the *output* size is 2048 dimensional (as used by the paper) since they mention their method saturates slower than other SSL methods. The paper uses three hidden layers for projection. I used one to reduce the training time. They suggest using projector output divided by 4 for the number of hidden units in the predictor, so I used 512 hidden units with output size 2048 [6]. For predictor, they use two layers, but I used one. They find that applying batch normalization to projector output slightly improves accuracy. I don't use batch normalization since the test accuracy is already 90%. They also mention that simple SGD works well for their architecture, so I used it for optimization [7].

In **Barlow twins** [Zbo+21] I use a projection head with 1 hidden layer (unlike the paper that uses 3) with 2048 nodes and output size 2048. The projection head also includes a batch normalization layer. To optimize, I used Adam.

For **supervised** training, I added a linear layer on top of the Resnet18 with ten outputs to calculate the logits and used Adam for optimization. The SSL models are unsupervised, so they need more epochs for the loss to converge. I have used either 800 or 400 epochs to train them to show that more training doesn't help with robustness. For the supervised model, 200 epochs gave an acceptable evaluation accuracy.

I add and fine-tune a final linear layer (readout layer) for SSL models to adapt the network to the labeled classification task. For the **standard fine-tuning of the readout layer** I trained it on the whole train set (without any augmentations), and the measurements in the plots are

[5]see the paper's appendix for CIFAR10 experiments.

[6]Note that projector and predictor are parts of the simsiam SSL training and have nothing to do with the readout layer.

[7]see the paper's appendix for their CIFAR10 experiments.

calculated on the CIFAR10 validation set. Using all the possible data gives the training method the best chance to learn a robust model. The backend (resnet18) weights are fixed while the last layer weights are trained. I used Adam with $lr = 10^{-2}$ and trained for five epochs and batch size 512. Note that this is a convex optimization problem. For the **adversarial training of the readout layer** the last layer is adversarially trained using my robustness library. I do apply augmentation on the train set here (random crop and horizontal flip) to make the models potentially more robust. The backend (resnet18) weights were fixed, while the last layer weights were trained using Adam and batch size 512 (learning rate is different for different models). Note that this optimization problem is not necessarily convex and is harder since the dataset adversarially adapts to the model as the weights are updated. In other words, the loss function (defined via samples) changes with the model weights. Due to this, adversarial training usually requires a learning rate smaller by two orders of magnitude for loss convergence.

A version of the supervised model where the last layer was separately trained like an SSL model is also included as "Linear separated supervised". I did this to make the supervised conditions as similar as possible to the SSL. I also tried using the labels in simCLR training and called it "supervised simCLR" for further comparison. In the supervised version of simclr, in each batch and for each image x, I sampled another image randomly from the same class (using the labels). I used it as x^+, and images from other classes are considered as negative samples in the Info-NCE loss notation. I also removed augmentation in supervised training to observe its effect on robustness; I called this model "supervised no-augmentation". A summary of the models and their specifications can be found in 1.

Finally, I evaluated all the models on the CIFAR10 validation set. The three main attacks I used for evaluation were FGSM, Apgd-ce, and Apgd-10. Apgd-10 is the same attack as the Apgd-ce, except I reduced the number of iterations from 100 to 10 so I can reduce the time for adversarial fine-tuning of the linear readout layer. Since APGD-ce seemed effective enough in decreasing model accuracies close to 0 and correlated well with APGD-dlr [8] I used it to compare the models. I have tried to keep the training and evaluation methods as similar as possible for all models. The standard accuracy of models on clean data can be seen in 13.

4.2 Results for Standard Fine-tuning the Readout Layer

Here for all the models, the readout layer was trained using standard supervised training (with no adversarial training). The evaluation results for different models using different attacks can be seen in 14. For all the attacks I have used l_∞ and $\epsilon = 8/255$ (similar to [Mad+19]). The values in the plot 14 are percentages. Both APGD attacks are from the Autoattack library with original parameters per [CH20a]. Initially, I trained simCLR and barlow twins for 800 epochs. However, since they had a similar performance (standard and adversarial) for 400 epochs, I trained the other two SSL methods (simsiam and simCLR supervised) for 400 epochs. Besides, in 13 we can see that these SSL models have standard accuracies higher or similar to the supervised model.

Observations from plot 13 and 14:

[8] APGD-dlr is another version of APGD where the loss being maximized is not cross-entropy, and in some cases, it might be more effective. see [CH20a] for more details.

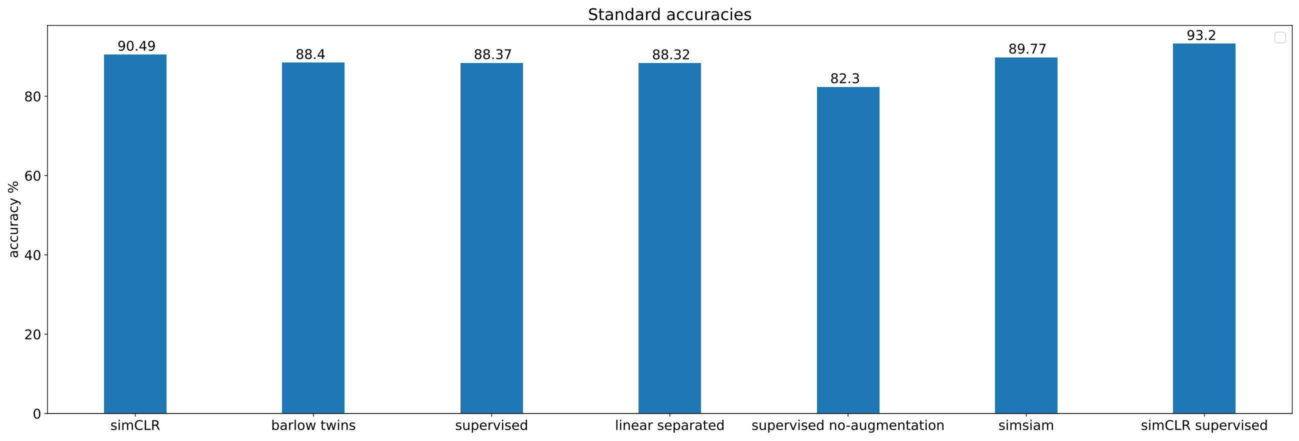
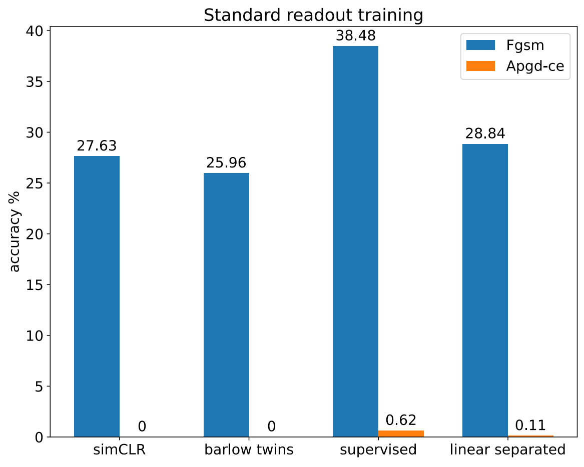


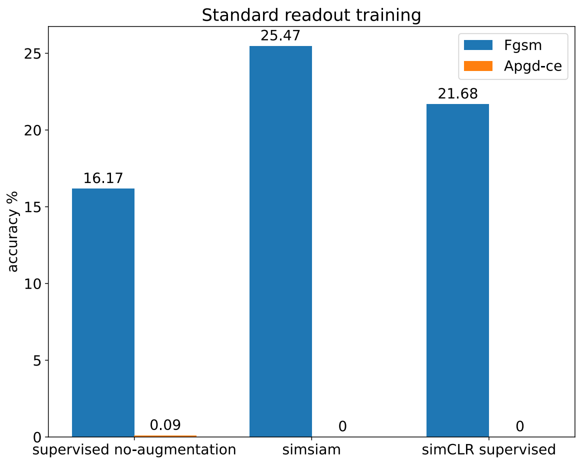
Figure 13: Clean accuracies

Model	Optimizer	Epochs	Loss
Simclr	LARS[YGG17]	800	$\ell_{i,j} = -\log \frac{\exp\left(\operatorname{sim}\left(\boldsymbol{z}_i, \boldsymbol{z}_j\right)/\tau\right)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp\left(\operatorname{sim}\left(\boldsymbol{z}_i, \boldsymbol{z}_k\right)/\tau\right)}$ (NT-Xent)
Simsiam	SGD	400	$\mathcal{L} = \frac{1}{2}\mathcal{D}\left(p_1, z_2\right) + \frac{1}{2}\mathcal{D}\left(p_2, z_1\right)$
Barlow twins	ADAM	800	$\mathcal{L}_{\mathcal{BT}} \triangleq \sum_i \left(1 - \mathcal{C}_{ii}\right)^2 + \lambda \sum_i \sum_{j \neq i} \mathcal{C}_{ij}^2$
Supervised	ADAM	200	$\ell_{i,j} = -\sum_i \mathbb{1}_{\{\hat{y}_i = y_i\}} \log\left(p_i\right)$ (Cross-entropy)

Table 1: The properties of different models used in the CIFAR10 experiments. Note variants of the same model (e.g simclr and supervised-simclr) have the same properties in this table. The simclr loss is also known as the Info-NCE or contrastive loss. The simclr and simsiam losses are for single samples and the final loss is the sum of all sample losses. For barlow twins and cross entropy loss the sum over samples is already included. For explanation of losses refer to 2.4 or their papers.



(a)



(b)

Figure 14: Robust accuracies for readout standard tuning. (a) simCLR and barlow twins trained for 800 epochs, supervised and linear separated for 200 epochs. Linear separated is a supervised model with readout layer fine tuned like an SSL model. (b) simsiam and simclr-supervised trained for 400 epochs, supervised no-augmentation trained for 200 epochs.

1. The SSL standard accuracies are on par with supervised models.
2. None of the models are robust. But the supervised model seems to be a little more resistant (0.62% robust accuracy for Apgd-ce).
3. Although FGSM reduces model accuracies significantly (50% or more), it grossly overestimates model robustness compared to the stronger Apgd-ce since (similar to all attacks) it approximates the solution to the adversarial problem. Therefore, FGSM is not suitable for making observations about individual models or comparing them.

I also observed that taking the best adversarial example of APGD-ce and APGD-dlr for each batch (setting version ='standard' in AutoAttack) results in an accuracy of 0 for all models. So none of the models have any adversarial robustness.

4.3 Results for Adversarial Fine-tuning the Readout Layer

In the previous section's experiments, we trained the readout layer in the standard way, and none of the models were robust. But what if the SSL models extract both robust and non-robust features (per [Ily+19] definitions)? however, we end up using shortcut/non-robust features since we train the readout layer standardly? To answer this question, I also trained the readout adversarially.

The readout layer was trained with APGD-ce (l_∞, $\epsilon = 8/255$, for 10 epochs) as the adversary since, as observed in the previous section, it seems to estimate robustness well. I use the implementation in torchattacks since the original implementation didn't let me change the number of iterations from 100 to 10. Changing the iterations was necessary since training just a single linear layer with 100 iterations of APGD-ce would take as long as training the backend! Besides, using 10 iterations is more realistic since models are usually trained using cheaper attacks and evaluated with stronger ones. I tuned the learning rate for Adam in the range $[10^{-6}, 10^{-3}]$ checking multiples of 10. The best learning rates were as follows:

1. Barlow twins: lr = 10^{-4}
2. SimCLR: lr = 10^{-4}
3. Linear separated supervised: lr = 10^{-5}

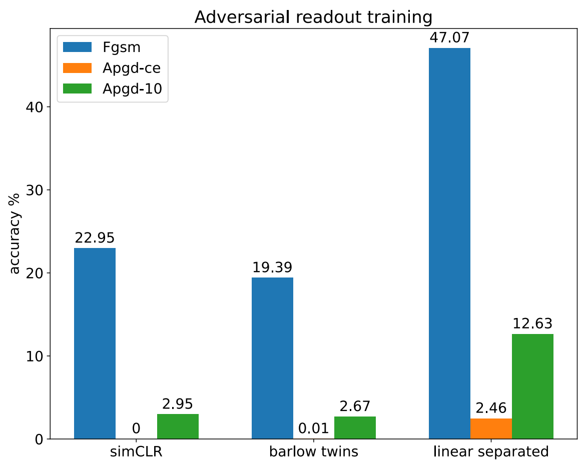
An interesting observation when training the readout layer adversarially using APGD-ce was that the adversarial loss could diverge (keep increasing) for smaller learning rates. This could be because the small learning rates (10^{-6}, 10^{-5}) cause underflow in floating point variables or other numeric instability problems. Usually, the adversarial loss increases or oscillates for the best learning rate and then decreases. As with standard training, the model with the best robust accuracy doesn't necessarily have the lowest loss.

Observations from results in 15:

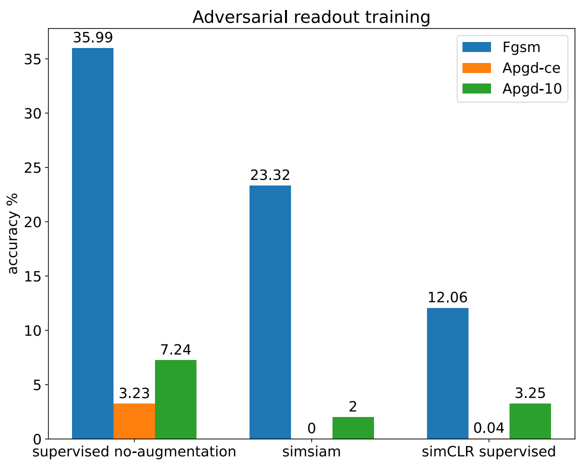
1. The best robust accuracy for APGD-10 and APGD is linear separated (supervised), which is the same as chance for APGD-10 and even lower for APGD.
2. All SSL models have almost no robustness. For instance, in the case of barlow twins, the 0.01 percent increase in the original APGD-ce is the equivalent of correctly classifying 1 sample in the validation set (the validation set includes 10^4 samples).
3. Note at this stage, even if we got some significant robustness in one of the SSL models, we couldn't necessarily say that the model is robust to all attacks since our results depend on the type of attack used in the evaluation. But it would have been interesting since we didn't use any attacks when training the backend network.

I also measured the average difference between the encoding of the Resnet18 (512 dimensional) before and after the attack $||(\textit{after} - \textit{before})||_2$, using APGD-ce($\epsilon = 8/255$, 100 iterations), for samples in the validation set [9]. Looking at the results in 2 and 3, we can see that the models with readout trained adversarially have a lower norm change compared to models with a standard readout layer. But when looking at individual models, the norm difference is not indicative of model performance. For instance, in 3, we can see that simclr has the lowest norm change, but it has 0% robust accuracy. It seems different training methods' norm differences are not comparable due to differences between the them. For example, barlow twins normalizes the latents (Resnet outputs) in its projection head while simsiam feeds the latents to a projector and a predictor. This means that different models use the latents differently and learn different latents. More importantly, for the norm difference to correlate with robust accuracy, we need to calculate it accurately, which is equivalent to calculating the Lipschitz constant of the network accurately. The Lipschitz calculation problem is hard, and the methods (similar to the attacks)

[9]Just because we use l_∞ norm for the attack doesn't mean we have to use the same norm for this calculation since the difference is calculated in the latent space of the images and not the image space itself. Also, note that l_1, l_2, and l_∞ norms are equivalent, i.e., each can be upper and lower bounded using the other two norms, so calculating one of them should be enough to observe any correlations with robust accuracy.



(a)



(b)

Figure 15: Robust accuracies for adversarially trained readout layer. (a) simCLR and barlow twins trained for 800 epochs, linear separated for 200 epochs. Linear separated is a supervised model with readout layer fine tuned like an SSL model. There is no "supervised" model in the plots since in order to train the readout layer adversarially I had to separate it. (b) simsiam and simclr-supervised trained for 400 epochs, supervised no-augmentation trained for 200 epochs.

give only approximate solutions; thus, inconsistent results are expected.

Model	Feature difference norm
Barlow twins	14.90
simCLR	17.82
standard supervised	11.02
Linear separated supervised	11.10

Table 2: Feature difference norms for standard readout training.

Model	Feature difference norm
Barlow twins	9.66
simCLR	8.50
Linear separated supervised	9.93

Table 3: Feature difference norms for adversarial readout training.

4.4 CIFAR10 conclusion

These experiments reject our hypothesis. In other words, using SSL models and losses doesn't automatically result in extracting more robust features from samples, and they seem to have lower robustness than supervised models. This either entails that the phenomena mentioned before (such as labeling noise or natural vs. human separation of data) do not significantly affect CIFAR10, or their benefits are offset by the tendency of SSL models to use easier non-robust features. Selecting a robust subset of features from SSL features by adversarially training the readout layer also fails with robust accuracies close to 0. In addition, the supervised model shows robust accuracies slightly higher than random prediction. This leads us to the second phase of my project.

5 Controlled datasets

The new question to answer is :

Why is the robust accuracy of SSL models lower compared to supervised models with the same standard accuracy ? and can we fix this?

To isolate the effect of the loss function, we need to rule out other confounders like the features present in the samples, dataset balance, model architecture and training method. Among these factors the dataset is the only one I haven't tried controlling or simplifying for our needs.

The problems observed in trying to learn robust features can be due to many factors such as redundant features or ambiguity of samples. For instance, as mentioned in 2.5 and 2.7.1 if there is an easy to learn feature such that the model can distinguish the images using them, then the model will use that feature and ignores others. (e.g. color instead of shapes or digits instead of complex shapes used by humans to classify). This is true for both supervised and self-supervised models , however this phenomena can affect SSL models more severely since it could lead to a deteriorating down stream performance since there are no labels used. Although this phenomena didn't manifest in our SSL models on CIFAR10 for *standard accuracy* (since the standard accuracies were similar to that of the supervised model), it can happen in other datasets such as STL-digits (STL images overlaid with MNIST digits) [Rob+21] and DigitOnImageNet (ImageNet images overlaid with MNIST digits) [CLL21b]. Thus, a similar phenomena could also be happening for the *adversarial robustness* of our SSL models.

The main issue is that we don't have control over any of these properties in real world datasets like CIFAR10, consequently we need to control the generating process of the data. By defining the causal factors we can control the features in images and other properties of the dataset. In this way we can isolate the effect of different contributing factors, therefore I decided to create a toy dataset. I can simplify the learning process and potentially gain some intuition regarding the phenomena affecting model robustness. An schematic of how causal factors can create a dataset can be seen in 11.

Previous works that analyze the effect of dataset on contrastive learning either assume a predefined data generation process [Zim+21] or they are not specifically designed for analyzing robustness of contrastive methods [Küg+21] (although I also experiment with 3dident) therefore, I decided to create my own toy dataset. It would be beneficial to keep the dataset and the classification task human understandable (e.g. simple image classification) since we can more easily visualize samples and run sanity checks. As depicted in 11 such a dataset would include:

1. The causal features (aka latent variables): these control the specific features that arise in a dataset and they provide the natural separation of the data or if continuous the spectrum of change. (e.g shape, color, texture)
2. Human labeling/perception: The labels humans accurately assign to the images. These might or might not align with the natural separation provided by causal features. Our hypothesis is that this can reduce model robustness due to arbitrarily contorted boundaries for classes.
3. Labeling noise: Humans can make mistakes while labeling. This is usually a small fraction

of the samples for good quality datasets.

To investigate the mismatch between causal features and human labeling we need causal features that are continuous or if discrete can be interpolated to create ambiguous samples (e.g. 3dident in 7). In addition I can try to reduce the number of pixels and channels in the images to reduce the dataset size 6.

6 Simple But Useful Toys

In order to control the effect of the dataset and model architecture on robustness and to study adversarial examples in a simple environment I decided to create a toy dataset. A toy dataset can eliminate phenomena like ambiguity (different classes overlapping in some regions) or concentration of measure [Mah+19] which can be removed if the model has 100% standard accuracy.

The idea is that if we have a dataset such that there exists a model (say given by an oracle) that is 100% robust (or close to 100%) ,**independent of the attack used**, we might be able to say that we have isolated the effect of the dataset and model architecture. A second model with the same architecture trained using the same data has the potential of being robust. This is unlike real datasets like CIFAR10 where there are no know universally robust models for classification. Note that CIFAR10 robust models are all robust with respect to a specific attack (usually trained via adversarial training) and non are 100% robust. Throughout this section in order to make calculations simpler I use l_∞ as the norm for adversarial perturbations.

6.1 The triangle square dataset

My first attempt at a toy dataset consisted of two classes: the "triangle" 17a and the "square" 17b. The 5x5 pixel patterns are placed in all possible positions in a 10x10 empty canvas where all the background pixels are set to 0 18a. Patterns always stay completely inside the canvas. This gives us 36 samples per class. The two patterns can be thought of as the causal factors for this dataset, the position of the pattern inside the canvas is the only other causal factor. We include all possible variations of the position factor in each dataset since we don't care about predicting it (it is an invariant for the prediction task). This helps the model generalize as best as possible on the in distribution samples (the total of 72 samples). Here we have a distribution with a finite set of samples (finite support).

Generally it is best to avoid having features specific to each class if they are not the target of our prediction since they could potentially act as a shortcut solution. These features need to be either fixed and equal for all classes or we need to include all possible variations for each class. In the same vein I made sure both classes have the same number of pixels set to 1 (shown in black) so that the model wouldn't use the shortcut of counting the pixels (eventually I observed that the model still uses a shortcut as explained later). For the same reason it is also not a good idea to give each pattern a distinct color. Coloring the shapes could be used if each class would uniformly sample from all possible colors but that would only complicate the dataset.

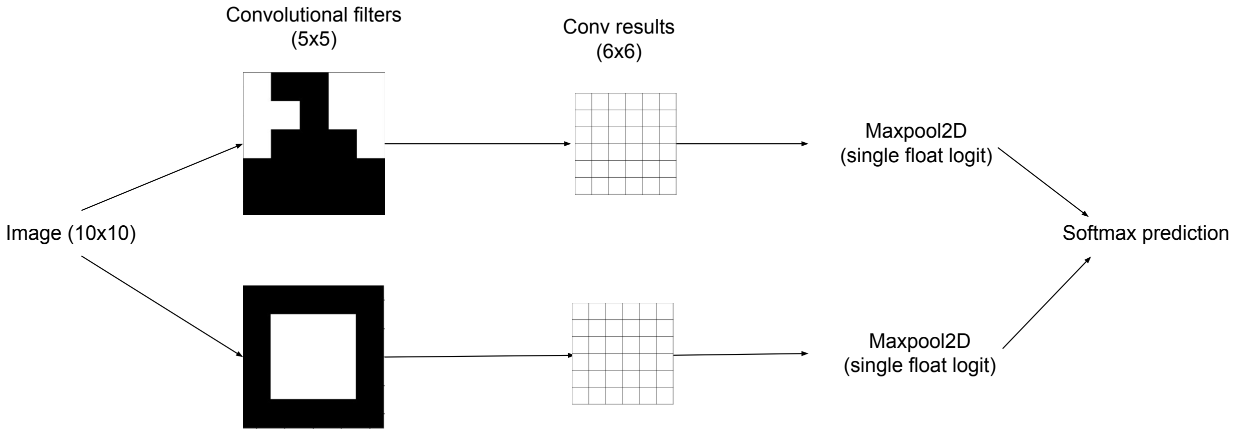


Figure 16: The hand made optimal projector.

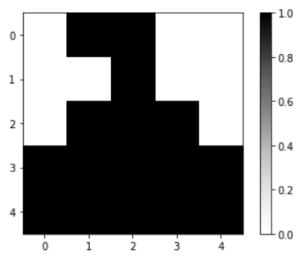
As a human we can play the role of the oracle mentioned before; one option for a 100% robust model is using one convolutional layer with two filters 16. Each filter is 5x5 and has the exact same values as the base patterns. I also divide the filter by the number of pixels set to 1 for normalization to make calculations simpler. This is followed by a max-pooling layer which gives two logits as output. The outputs are 2 floating point values corresponding to the two classes and the larger one is the prediction of the network (or equivalently we can calculate softmax probabilities) [10].

We can easily prove that this model is 100% robust to any $l_{\inf}$ perturbation with norm $\epsilon < 0.5$ independent of the attack. Intuitively if an optimal attack wants to change one class to another it should take advantage of the most pixel overlap between the two patterns. For this dataset this happens when the two patterns are exactly on top of each other. The optimal attack should also use the maximum perturbation magnitude allowed since the convolutional filters are linear so for instance to change a triangle to a square the optimal adversarial image will look like this:

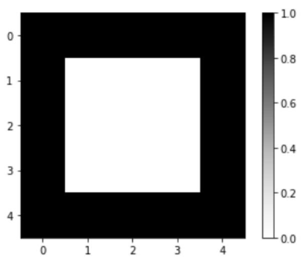
$$\textit{Original triangle pattern} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

$$\textit{Adversarial pattern} = \begin{bmatrix} 0+\epsilon & 1 & 1 & 0+\epsilon & 0+\epsilon \\ 0+\epsilon & 0 & 1-\epsilon & 0 & 0+\epsilon \\ 0+\epsilon & 1-\epsilon & 1-\epsilon & 1-\epsilon & 0+\epsilon \\ 1 & 1-\epsilon & 1-\epsilon & 1-\epsilon & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

[10]This function is non-linear but convex since the outputs are the results of taking the maximum of a group of linear functions. I think using binary cross entropy loss we would also get a convex optimization problem.

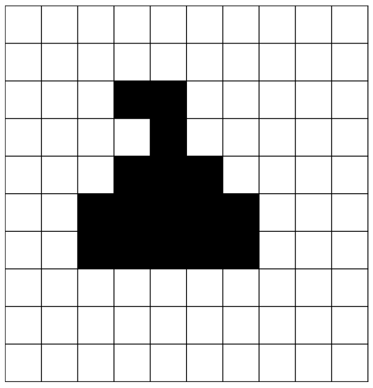


(a) Triangle pattern



(b) Square pattern

Figure 17: The basic patterns


(a) An example of the triangle pattern on the canvas.

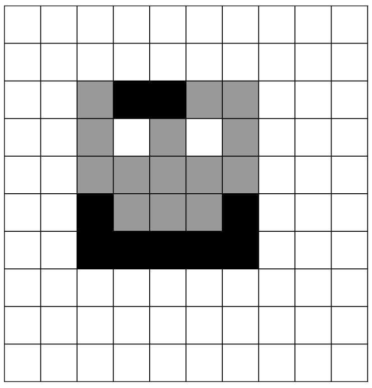

(b) An ambiguous image.

Figure 18: Instances of images. The ambiguous image is not in the dataset.

Note the adversarial pattern will replace the original pattern on the canvas. For $0.5 \leq \epsilon$ The samples will become ambiguous as shown in 18b and it is impossible to predict the class of the original image. Obviously our handmade model also has 100% standard accuracy. Note that due to the max-pooling layer this model is not linear but it is convex. Obviously the robust model is not unique; for instance, we could drop one of the filters (or choose its values randomly) and still get a robust model. Since we only have 2 classes and assume all inputs are either standard samples or their perturbations. [11].

I then trained a model with the exact same architecture as the robust model on this dataset to see if it would be robust. We can see the filters resulting from standard training in 19. The model trained for 20 epochs has standard accuracy of 100%. The robust accuracies using PGD($\alpha = \epsilon/2$, steps=10) are shown in 4.

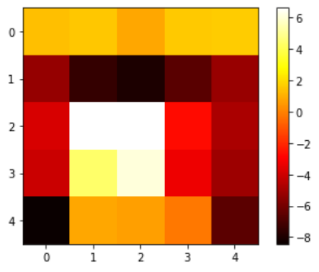
I also adversarially trained the model with FGSM($\epsilon = 0.1$) and the filters are shown 20. We can see that the first filter corresponding to the "triangle" class already looks more similar to the actual shape.

One potential idea to create a robust dataset might be to remove non-robust features by

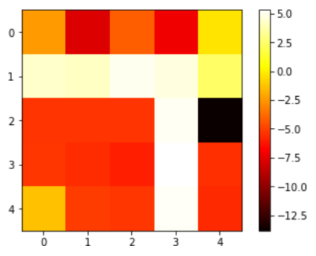
[11]Generally for any dataset without adversarial ambiguity (without class overlaps when considering all samples within ϵ neighborhood of the original class boundaries according to some norm) for an n-class classification task we only need to robustly detect n-1 of the classes.

ϵ	Adversarial accuracy (%)
0.1	99
0.2	54
0.3	11
0.4	0

Table 4: The robustness of the model standardly trained on the triangle-square dataset for different values of ϵ.

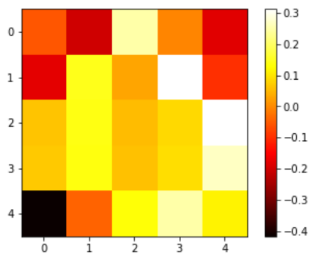


(a) triangle

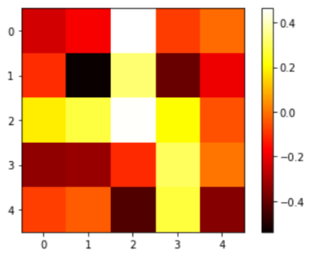


(b) square

Figure 19: The standard training filters.



(a) triangle



(b) square

Figure 20: The adversarial training filters (FGSM). An interesting fact is that when I used perturbations bigger than 0.1, gradient descent was very unstable and could no longer optimize the loss effectively even when decreasing the learning rate by 4 orders of magnitude. This shows that adversarial training might have issues with big perturbation values since the loss landscape shifts significantly with each epoch which makes converging harder.

modifying the image pixels based on saliency maps or other similar pixel level methods. In this example we can see that the network can pick up on features that are a subset of the correct pattern but it's not detecting the whole pattern. Such as in 19a where the network pays a lot of attention to center pixels but not the whole "triangle" shape. Then removing or weakening these pixels by adding noise or penalizing model for using them also removes the general pattern which was the robust feature. So **"non-robust" features can also be a subset of robust features**. In contrast in other works such as [Mad+19] they optimize the representations of the images using the latent space of a robust model to create a robust dataset.

6.2 The L-T dataset

There is still an important unanswered question:

Is it possible to train a robust model using this dataset alone ?

when we created the optimal model by hand we used our human biases which helps us create the robust model while the learning algorithm doesn't have that possibility. This results in standard training yielding a model that doesn't correctly detect the shapes. So to truly control the dataset we need a new definition: **A robust dataset** is a dataset such that there exists a learning method (optimization method) and an architecture such that when the learning method is used to train the model using the provided data (without using any extra data) the model is robust.

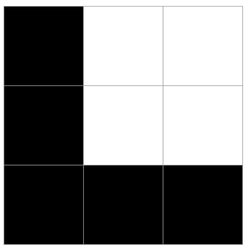
Since we are in a toy example the learning method can be computationally expensive or even theoretical. One possible example using brute force is this: for each of the 36 samples add all images that are in its ϵ neighborhood to the dataset, then train the model on this new dataset. We know that all values in a computer are finite precision so this dataset is finite. We can make this method computationally feasible by reducing the size of images and using lower precision pixels instead of the standard 64-bit floating points.

What kind of model will we get if we train our model standardly on this new dataset ? will it be 100% robust like the handmade model above? how does it compare to adversarial training ?

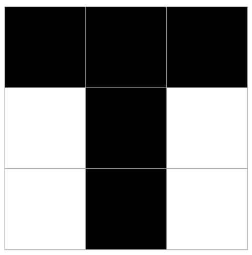
One computationally efficient way of doing this is to reduce the precision of the pixels. For example assume we divide the interval between 0 and 1 into 5 segments so we would get 6 distinct values for each pixel $\{0, 1/5, 2/5, ..., 1\}$ so the smallest possible pixel value increments/decrements is $1/5$ (excluding the case of not changing the pixel at all). Also assume l_∞ norm and $\epsilon = 2/5$ then for each 10x10 image, assuming norm infinity, there are 5 possible modifications for each pixel $\pm 1/5$, $\pm 2/5$ and 0. Thus there are 5^{100} neighbors in the ball centered at an original image. So the dataset size is $N * 4^{100}$ where N is the number of standard images. This gives us $72 * 5^{100} \approx 5.68 * 10^{71}$ images for the triangle-square dataset. We can see that even for the toy dataset above ($N = 72$) the memory requirement can get astronomical. Even if we exclude the samples inside the epsilon ball (images such that all pixels only use $\pm 1/5$ or 0 so the total count is $72 * 3^{100}$) and only consider the ones on the boundary we would get $72 * 5^{100} - 72 * 3^{100} = 5.679798518 * 10^{71}$ samples which is essentially the same number. Due to the curse of dimensionality in higher dimensions excluding images inside the center of the ball wont decrease the dataset size significantly. We can also show the same concept in continuous space, in a hypercube as the number of dimensions increases most of the volume of the cube concentrates close to the boundary of the cube [12].

To reduce the size of the dataset we need to decrease the problematic term 5^{100} which is the possible options for each pixel to the power of number of pixels in a single image. So we need to decrease both the image resolution and the options for each pixel. To this end I tried a 4x4 canvas and 3x3 patterns. The two "L" and "T" patterns are shown in 21. I used a model architecture similar to 16 adapting the sizes of filters for the new dataset. Initially I evaluated the model trained on the standard dataset to show that it uses shortcut solutions. I trained the

[12] see a proof here from the math department of UCdavis.

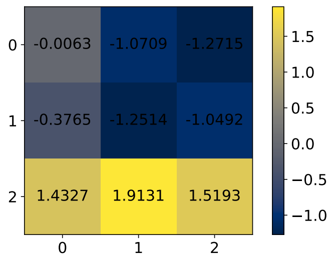


(a) The L pattern

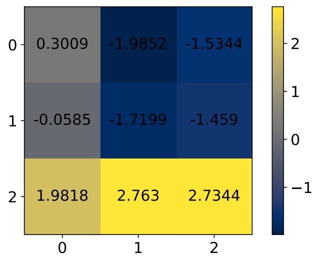


(b) The T pattern

Figure 21: A simpler dataset



(a) trained for 20 epochs



(b) trained for 200 epochs

Figure 22: L filter results for standard model training using standard data (8 images)

model using Adam($lr = 0.1$) and used PGD($\alpha = \epsilon/2$, steps=10) to calculate robust accuracy. As before the model didn't learn robust filters but the adversarial accuracy only drops for $0.3 < \epsilon$. The robust accuracy for $\epsilon = 0.4$ is 12% for both 20 and 200 epochs of training, indicating that the model has converged and is using a shortcut solution 22.

6.2.1 Simplifying assumptions

In order to further decrease the dataset size I made some assumptions:

Assumption 1: Any adversarial image (incorrectly classified by *my* model architecture) that uses a perturbation smaller than the maximum perturbation ($Attack$-ϵ) for any pixel can be replaced by an image that uses the maximum perturbation for all the modified pixels such that, the new image is also classified incorrectly and the logit difference is even worse than the first image.

This assumption enables me to only consider the cases were the pixels are modified by the maximum perturbation possible. For l_∞ norm this corresponds to only considering the corners of the hypercube and dropping the points on the sides or inside the cube. Intuitively (for my specific model) if the attack needs to change one pattern to another (change the class) there is no reason for the attack to only use a fraction of the perturbation allowed since: **1.** The model we use is *almost* linear (2 conv filters followed by max pooling) this means that the model separates the classes using hyperplanes so if a point belonging to the epsilon neighborhood hypercube is on the wrong side of one of the boundaries then one of the corners must also be misclassified. Conversely if all of the corners of the hypercube are on the correct side then the whole hypercube is on the correct side (using a 3D intuition). **2.** the target patterns (standard images) use either 0 or 1 pixels so the attack has the best chance at fooling the model if it uses all of the perturbation allowed. As mentioned before we need to have $\epsilon < 0.5$ so the adversarial samples wont be ambiguous. When running an attack the attack might perturb some pixels less than the maximum amount as it is performing gradient ascent. The point is that the position that corresponds to the maximum value calculated by maxpool layer (the final decision of the model) for the adversarial class will be definitely using the maximum perturbation possible.

Assumption 2: For my *almost* linear model and l_∞ norm (which means the ϵ-neighborhood is a hypercube) and assuming the above intuition is correct, we can see that we don't need to discretized the $[0, 1]$ interval since the above argument is independent of the granularity of discretization. So it would be enough to have 3 options for each pixel $\{-\epsilon, 0, +\epsilon\}$. Note that for l_2 norm this doesn't really help reduce the new dataset size since every point on a hypersphere is a "corner" (uses maximum perturbation) unless we fit the hypersphere in a hypercube.

Assumption 3: Furthermore, I have observed that non of the attack samples leave the [0,1] interval for each pixel so there is no reason to consider values out of this range. So for a 0 pixel I only considered the options $\{0, +\epsilon\}$ and for a 1 pixel $\{1 - \epsilon, 1\}$ giving me 2 options per pixel I call these **in-boundary neighbours** (this assumption turned out to be incorrect as explained later).

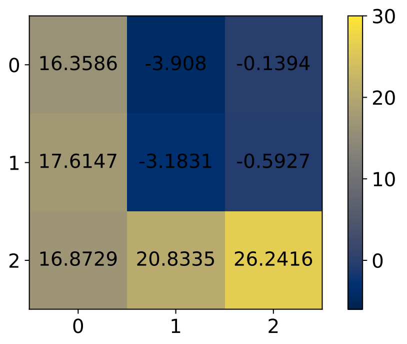
Assumption 4: Moreover I only apply the perturbations to the pixels corresponding to the location of the pattern on canvas and not the whole canvas to decrease the exponential term further. This corresponds to the adversarial class's pattern lying exactly on the correct pattern location. This assumption was also wrong as shown in a counter example in 24, but there is a correct version of the statement. The adversary should try to get as much pixel overlap (matching 0 and 1 values) between the initial pattern and the target pattern to maximize the target pattern's convolutional filter output with the *limited budget* ($\epsilon < 0.5$) it has. This doesn't necessarily correspond to putting the two patterns on top of each other ! It doesn't make sense for the adversary to prefer non-overlap pixels to overlap pixels since it would have to for instance, get a 0 pixel as close as possible to a 1 pixel (which is $0 + \epsilon$) while it could have used a 1 pixel that already exists (this is left as a future effort to optimize the dataset further.)

6.2.2 Proof of model robustness

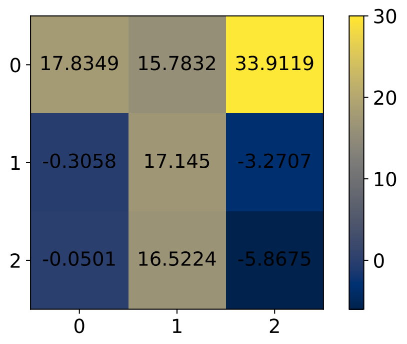
Before getting to the proof it is important to note that there are 2 epsilons: $Dataset$-ϵ which is used to create the samples on the boundary and should be the bigger epsilon to train a provably robust model and $Attack$-ϵ which is the attack budget. I will fix $Dataset\text{-}\epsilon = 0.41$ and $Attack\text{-}\epsilon = 0.4$ for all models mentioned in the rest of this section unless another value is specified. Using assumption 1, assuming l_∞ and $Attack\text{-}\epsilon < Dataset\text{-}\epsilon < 0.5$ for the attack and the fact that the dataset includes all possible $Dataset$-ϵ perturbations we can show that:

Theorem: For a model with 100% standard accuracy on the *extended* dataset, no adversarial perturbation for the images in the *standard* dataset can change the model's predicted label.

Proof: By contradiction assume there is such an adversarial example, which uses $Attack$-ϵ perturbation. then we can create a corresponding sample which uses the same perturbation



(a) L filter



(b) T filter

Figure 23: The filters learned by the robust model.

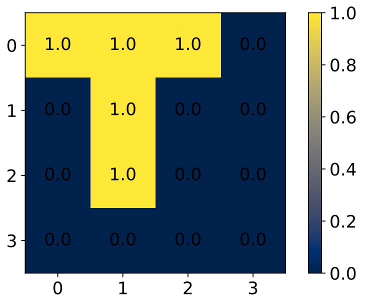
pattern but with magnitude $Dataset\text{-}\epsilon$ and the model also misclassifies the new image. But the new image is part of the extended dataset and the model correctly classifies it. □

I will also experimentally show the correctness of the theorem using a PGD attack later. According to the above assumptions for two 3x3 patterns moving on a 4x4 canvas I need to modify all the canvas pixels using $\{-Dataset\text{-}\epsilon, 0, Dataset\text{-}\epsilon\}$ options. This gives a dataset size of $3^{16} * 8 = 344,373,768$, I will call this the **extended dataset**. For the experiments mentioned below I also attempted pixel normalization using mean and std of all pixels in the dataset. Normalization didn't seem to affect any of the robustness results mentioned below, only slightly improving some of the models [13]. So I haven't used normalization unless specified.

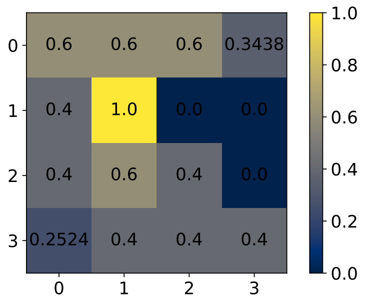
Standardly training my model on the **extended** dataset leads to 100% standard accuracy on the **extended** dataset and 100% robust accuracy on the **standard** dataset. I used Adam($lr = 0.1$) and trained for $epochs = 6$. The learned filter can be seen in 23. Since the dataset was too big to fit in the GPU memory unlike before I used batch gradient descent[14]. Note that any set of weights for my model that has 100% standard accuracy on the **extended** dataset but a robust accuracy less than 100% on the **standard** dataset using any attack is a counter example and that is how I detect my wrong assumptions.

The model trained using the wrong version of assumptions 3 and 4 (only inbound perturbations and only perturbing the pixels in the pattern's location) with Adam($lr = 0.1$) and $epochs = 200$ reaches 100% standard accuracy on the extended dataset but shows a robustness of 37% (5 samples misclassified out of 8 in the standard dataset) and standard cross entropy $loss = 0.0078$ which yields a counter example. When I removed assumption 3 but still used the wrong version of assumption 4 (only perturbing the pixels in the pattern's location) the counter example 24 was produced by PGD(attack_epsilon=0.4, alpha=0.04, steps=50). Even though the model is correctly learning the shapes in 25 it is still not enough to make the model robust! meaning the relative values of the weights matter.

[13]Which makes sense considering the range of values we use is close to the interval $[-1, 1]$.
[14]refer to code for more detail

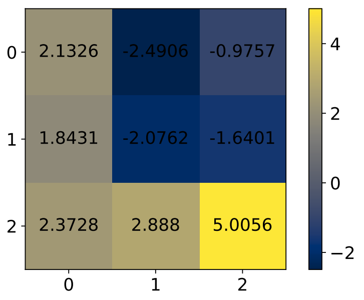


(a) a standard T image

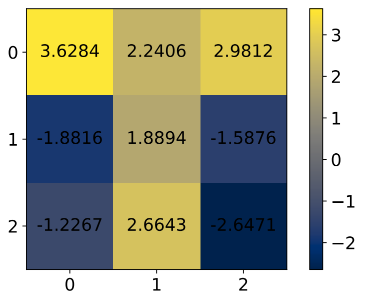


(b) corresponding adversarial image found by PGD

Figure 24: A counter example as calculated by PGD



(a) The L filter



(b) The T filter

Figure 25: The filters learned by the model using assumption 4.

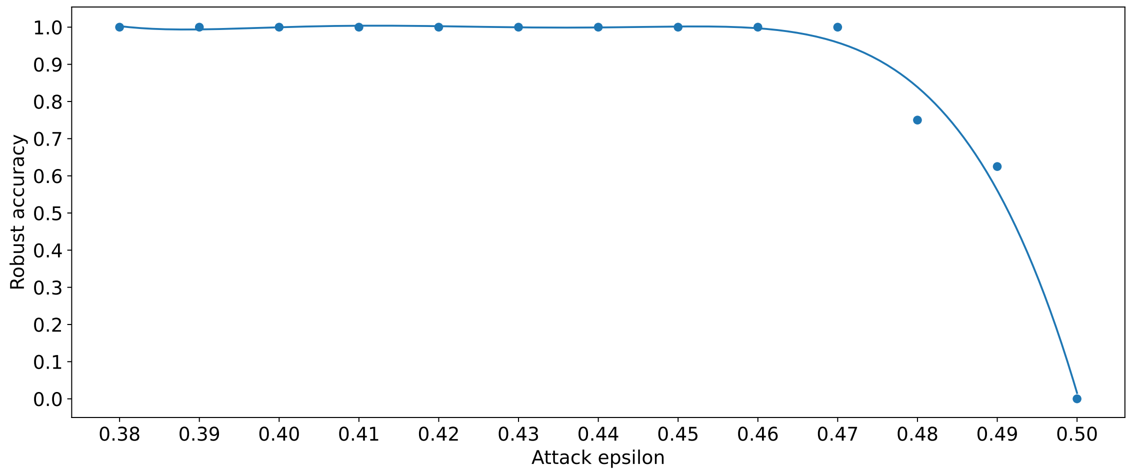


Figure 26: The robust model maintains 100% robustness even when $Dataset\text{-}\epsilon < Attack\text{-}\epsilon$. The $Dataset\text{-}\epsilon = 4.1$ is fixed. The dots show the values calculated by PGD the curve is regressed using a polynomial of degree 5. We see that the model maintains robustness even after $Dataset\text{-}\epsilon < Attack\text{-}\epsilon$. It should be the case that for $Attack\text{-}\epsilon = 0.5$ we get 0 accuracy since the samples become ambiguous. I also tried using PGD with $\alpha = 0.4 * 0.1$ fixed and the results were the same.

6.2.3 Observations from the Triangle-Square and L-T datasets

As empirically observed in [Mad+19] a robust model needs more capacity (i.e. number of weights and nodes in a neural network) than a standard model. This is also confirmed in 19 where the standard model learns only a subset of the shape meaning a model with smaller convolutional filter could have also separated standard images correctly. Obviously as shown in 19 using a higher capacity model wont guarantee a robust model but it can be a necessary condition if the initial model is too small.

The L-T dataset shows that, with the assumptions made above regarding the dataset, it possible to train a model robustly (100% robust accuracy) using standard training and the cross-entropy loss. Note we don't use any attacks during training and as I proved above the model is 100% robust to all samples in the ϵ-ball which means **our model is robust against any attack.** In addition, even when I use $Attack\text{-}\epsilon$ values bigger than $Dataset\text{-}\epsilon$ the model is still robust against PGD(alpha=$\epsilon * 0.1$, steps=50) 26. Furthermore there is no sudden drop in robustness after $Attack\text{-}\epsilon$ surpasses $Dataset\text{-}\epsilon$ which is further proof that the model has not overfitted any specific hyper-parameters during training 27. This is unlike adversarially trained models mentioned in [Mad+19] figure 6 (a,c).

Plot 27 shows that only adding more data is not enough to increase model robustness and we need the data points on the boundary otherwise as we decrease the number of samples on the boundary in our dataset or alternatively as the samples get further away from the boundary the model loses robustness (here we actually increase the attack budget which means we expand the hypercube but it a similar scenario). 26 shows that if we use a non-ambiguous extended

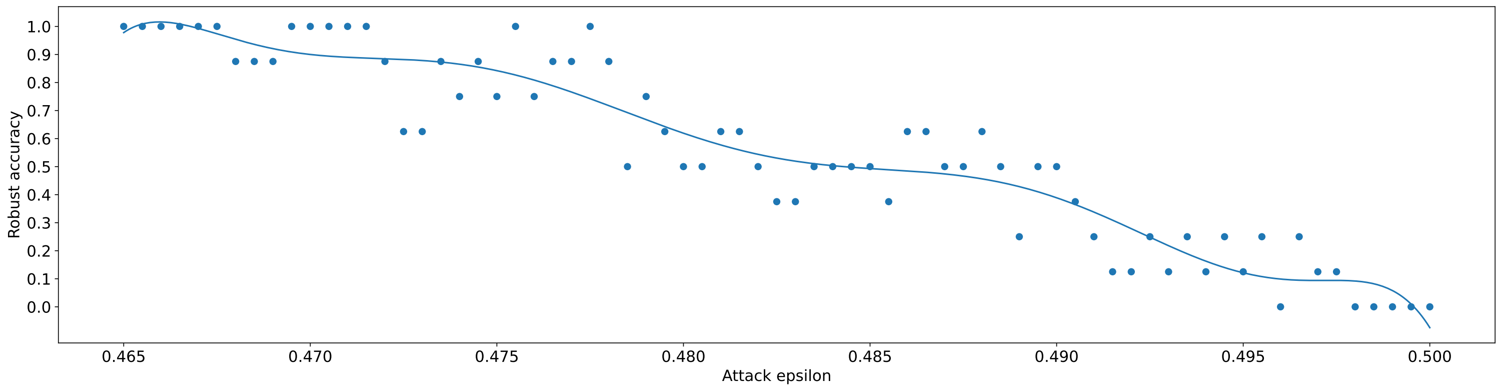


Figure 27: Evaluating robustness with more $Attack\text{-}\epsilon$ precision (zooming in on the 26 plot) after the robust model starts losing accuracy ($0.46 \leq Attack\text{-}\epsilon$). The $Dataset\text{-}\epsilon = 4.1$ is fixed. The dots show the values calculated by PGD the curve is regressed using a polynomial of degree 8. We can see the drop in accuracy is slow.

dataset it is possible to learn a robust model using the standard Cross-entropy loss and Gradient descent even providing robustness further than expected i.e. **The usual elements used in a deep learning pipeline are capable of training a robust model**.

We can conclude that an important part of training a robust model is having access to such a dataset (specially if we also prove that including the boundary samples in the dataset is crucial as empirically shown in 27). This is also in agreement with the assumption of statistical learning regarding having an i.i.d sample of data which is usually ignored in an adversarial setting. To my knowledge, this is the first time that a standard training pipeline has been used to train a model that is 100% robust to all adversaries without relying on any attacks or other modifications of the deep learning pipeline other than the dataset.

Another observation regarding the dataset extension method is that we have created a new dataset consisting of all the samples in the ϵ neighborhood, so this still doesn't show that its possible to learn a robust model from our original dataset with only 8 samples! Obviously this is an upper bound to the number of samples needed to train a **100%** robust model **independent** of the attack (an astronomical number if the pattern size [15] is more than 5x5. [16]). **Can we show this upper bound is tight ?** In this toy example it might be possible to show that:

1. Hypothesis: we need to include all the samples on the boundary [17] to train a robust model using gradient descent. If one of the boundary samples is not in the training set the model boundary could be non-robust as shown in 28(b). Plot 27 (where I increase $Attack\text{-}\epsilon$) might add

[15]or the canvas size for any image dataset not just one that classifies patterns.

[16]For instance, a canvas or pattern of size 5x5 pixels and with the standard dataset having a size of 8 the extended dataset will have $3^{25} * 8 \approx 6.8 * 10^{12}$ images

[17]As we have seen including all the boundary samples might not be necessary since my robust model was also robust for $Dataset\text{-}\epsilon < Attack\text{-}\epsilon$ 26. In that case we can replace "all the samples on the boundary" with "A set of minimally sufficient samples that are sufficient to make the model robust". However I doubt without adding additional assumptions about the type of neighborhood or the model we would be able to do that.

validity to this hypothesis, yet it is not the same setting since I don't remove a neighbour but increase the neighborhood size for all standard samples. My intuition regarding this hypothesis in 2D is shown in 28.

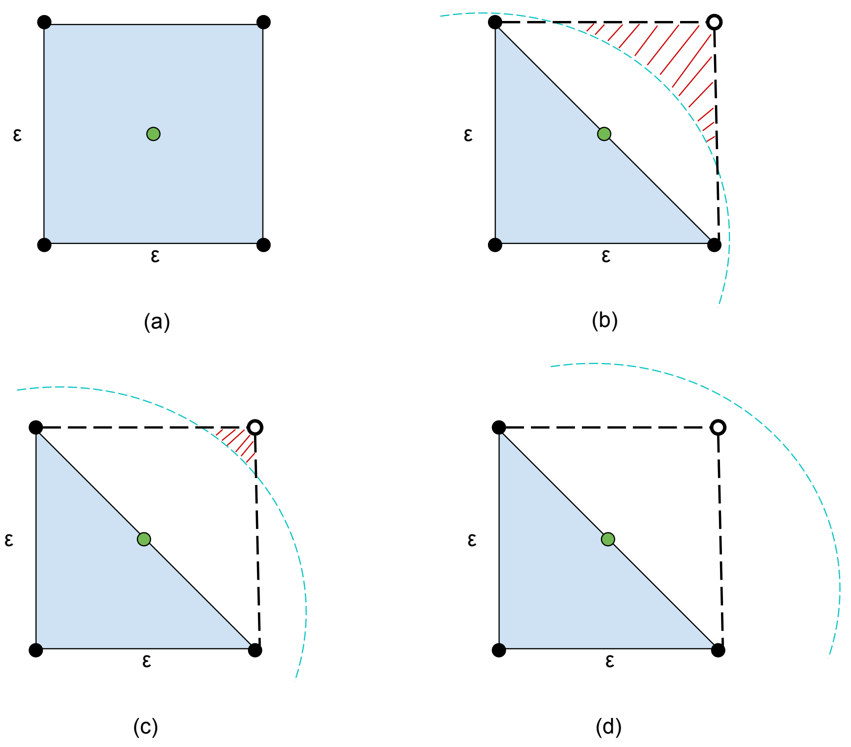


Figure 28: An example for excluding a boundary sample in 2D for my hypothesis, here I assumed the adversary uses l_∞, so the neighborhood is a square. (a) The standard sample (shown in green) centered at l_∞ ball of size ϵ. (b) A general model (possibly non-linear and non-convex) standardly trained using the extended dataset but excluding a single corner sample shown as a white dot on the top right. The red hatched area shows the region where the adversary can sample to fool the model (a.k.a the vulnerability region) note the diagonal line could be part of the boundary of a model that has 100% standard accuracy but is still non-robust. (c) Adversarially training the model, the adversary produces the samples in the vulnerability region, which makes it smaller (i.e. the model is more robust). However since the adversarial problem 1 is non-convex there is no guarantee we would include the excluded neighbour, so we still have a vulnerability region. This is similar to how adding random noise (i.e. sampling random points in the neighborhood) wont make a model more robust. Obviously adversarial training is a much better sampling method but it doesn't guarantee a fully robust model and it overfits the attack used. (d) Including all the boundary points in the dataset puts the model boundary in the desired region which eliminates the vulnerability region. Note for a general (possibly non-convex and non-linear) model this means also including the samples on the edges of the square and including the 4 corners might not be enough !

2. Fact: The number of non-boundary samples (samples inside the ball) is insignificant compared to boundary samples. This is already shown in 6.2 as a numeric example. This means most of the training happens on the boundary samples so their existence is crucial to model robustness. This is also confirmed in the L-T dataset by the fact that we only train the model on boundary images and standard images. Additionally in the L-T dataset excluding the standard images (corresponding to the 0 change option for the pixels) still gives us a 100% robust model. Consequently, the minimal options for each pixel to train a robust model are: $\{-\epsilon, \epsilon\}$.

This would mean that even for a toy dataset in order to get a **100%** robust model **independent** of the attack we would need an astronomical number of samples if we use 64-bit precision pixels. Note that with adversarial training we can obviously reach 80% or 90% robustness **for a specific type of attack** but as papers have experimentally shown the model fails to perform equally well for a stronger attack and thus the endless arms race ! This dataset could show a limitation of our neural network models trained using gradient descent compared to a more robust model like a human brain which obviously doesn't need an astronomical number of samples to be robust ! [18]

7 Knowing the causes is not enough ! (Causal3DIdent experiments)

The previous toy datasets only used simple basic pixel patterns as the causal factor to create a small dataset. Although they provided the possibility of running computational methods that are infeasible for real image datasets, their features didn't represent the complexity of real-world image classification tasks. To investigate further I used the Causal3DIdent dataset (a.k.a 3dident) from [Küg+21]. Compared to the toy datasets Causal3DIdent is much closer to real world images which can lead to better insights regarding the factors that affect model robustness on real datasets. Causal3DIdent's causal graph is shown in 29, each independent causal factor on the top row controls different features of the scene (e.g spotlight position or object-class) which can either be considered as the feature of interest for the model to regress/predict or contribute to the calculation of other dependent factors (e.g. object-position). This causal structure would give us the appropriate controls needed to experiment with different hypotheses using the dataset. There is real and rendered versions of the dataset but only the rendered version is available.

7.1 Dataset creation

The dataset is created using 4 main causal factors: spotlight position (pos_{spl}), hue (hue_{spl}), object class and background hue (hue_{bg}). These variables are all sampled independently form $U[-1, 1]$. All object variables are dependent on these initial variables by changing the mean of a normal distribution (for more details see appendix B of [Küg+21]). In total there are 7 object classes: Teapot, Hare, Dragon, Cow, Armadillo, Horse and Head. The images are 224 × 224 pixels and are created using blender. Each object is rendered using the following attributes:

1. $x \in [-3, 3]$
2. $y \in [-3, 3]$
3. $z \in [-3, 3]$
4. $\alpha \in [0, 2\pi]$
5. $\beta \in [0, 2\pi]$
6. $\gamma \in [0, 2\pi]$
7. $\theta \in [0, \pi]$
8. hue object $\in [0, 2\pi]$
9. hue spotlight $\in [0, 2\pi]$

[18] I believe that we are fooled by comparing artificial neural networks' robustness to that of the brain and assuming it is a simple task since 1. The two have almost nothing in common (in the way they are trained or the model they use) 2. It might also be possible to attack the visual cortex with l_∞ and small epsilon but since we don't have a model for the visual cortex we can't design any tailor made attacks.

(x, y, z) are object the position, (α, β, γ) are the object rotation and θ is spotlight rotation. There is also hue background in $[0, 2\pi]$ which is changed per scene.

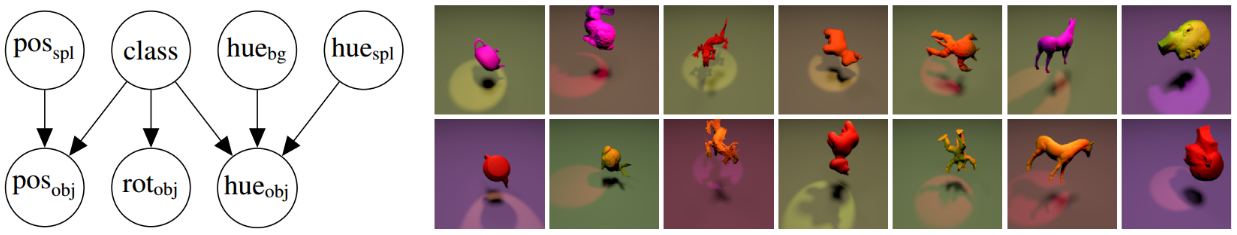


Figure 29: (Left) Causal graph for the Causal3DIdent dataset. (Right) Two samples from each object class.

7.2 General idea

Up until this point we only considered classification tasks where there is no objective meaningful metric defined for the distance between classes. But in 3dident we have the opportunity to regress a continuous random variable that is a causal factor for creating images. We can then divide this variable into separate regions (according to the specific hypothesis we want to test) to create discrete classes. For instance we can consider the spotlight rotation and discretize it into 3 non-overlapping classes as in 30 (the spotlight is rotated on a semicircle above the object see 7.3 for details). I then define the parameter δ^* as the maximum perturbation that can be tolerated by the *classification task* before values from one class end up in a region belonging to another class as shown in 30. Note that δ^* is purely a property of the classification task and has nothing to do with the network architecture, loss or the optimization method used to train it. This setup makes it possible to separate δ^* from the model robustness. In case of an inherently discrete object classification task (e.g. cat vs dog) one can think of estimating the output probabilities as a regression task but there is no well defined distance or similarity metric, so it is hard to calculate the δ^* for such a dataset.

Discretizing a regression task into a classification task gives us much more control over how the classes are defined, whether they are ambiguous (the classes overlap in the regression space) or well separated (like in 30) and how many classes we have. This also makes characterizing adversarial examples simpler. Furthermore the model can be divided into two parts: The network that regresses the continuous variable and a classification head added on top. Note that the classification head is only a function of the classification task we define and doesn't need to be trained. To provably determine the amount of perturbation this classifier is robust to we only need δ^* and the lipschitz constant of the network λ.

Assuming the network is λ-lipschitz and the input is x and the bounded perturbation vector is r s.t. $\|r\| \leq \epsilon$ making the adversarial image $x + r$ we have:

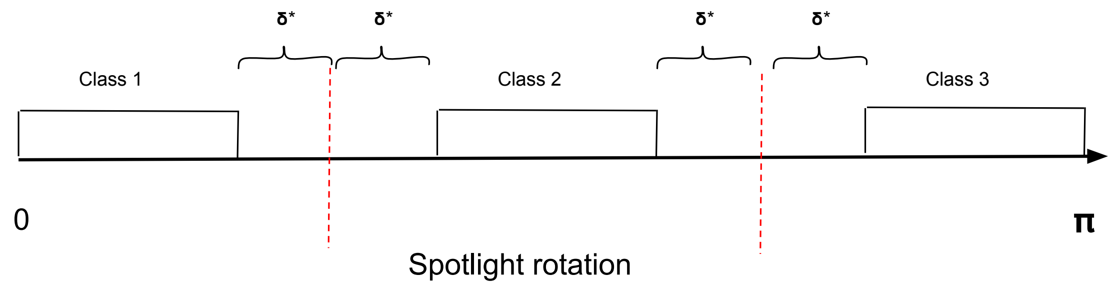


Figure 30: The spotlight rotation is divided into 3 non-overlapping classes. Each class has the same range and the gaps between classes are equal in length. Spotlight rotation is sampled uniformly from the whole range $[0, \pi]$ so the classes are balanced. Since the classes don't overlap there exists a model that is robust. The most robust classifier would separate the classes using the middle point of gaps (red dashed lines). Therefore δ^* is the maximum perturbation that can be tolerated by this classification task.

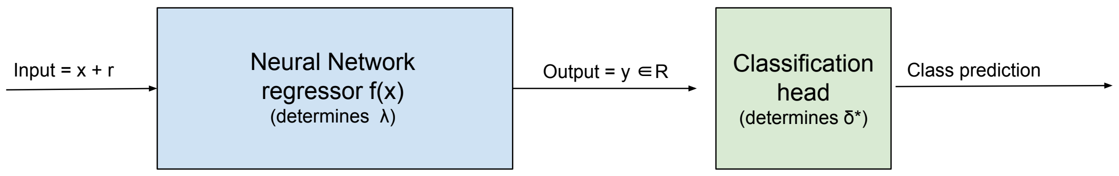

Input = x + r
Neural Network regressor f(x) (determines λ)
Output = y ∈R
Classification head (determines δ*)
Class prediction

Figure 31: The classifier including the regressor and the classification head. Note that the classification head is purely defined by our definition of classes and is not learned.

$$\forall x \in \mathbb{R}^n \quad f(x) \in \mathbb{R} \tag{20}$$

$$\forall x, r \in \mathbb{R}^n \quad |f(x) - f(x+r)| \le \lambda \|r\| \le \lambda\epsilon \tag{21}$$

$$\lambda\epsilon \le \delta^* \tag{22}$$

The most perturbation that the classification head input can tolerate, and for the sample not to pass the classification boundaries (red lines in 30) is δ^* thus the third line in the equations above. Given δ^* and λ of a specific network we can provably say that our classifier is 100% robust against any attack where the perturbation is bounded as $\epsilon < \frac{\delta^*}{\lambda}$. Similarly given δ^* and ϵ we can calculate an upper bound for the lipschitz constant of the network required for the whole model to be 100% robust in this classification task [19].

This method could also be generalized to a regression network with a multidimensional output. The practical issue with using this setting is finding a method to train the network such that its lipschitz constant is bounded by a specific value λ. Usually even estimating the lipschitz constant of a network can be computationally expensive. However, if we manage to train such a network for the regression task then we would have a 100% robust model for the classification

[19]We can also use $\lambda\epsilon \le \delta^*$ to estimate the lipschitz constant of the network (upper bound it), fixing the δ^* we keep decreasing ϵ until the robust classification accuracy is 100%. The more points sampled from the data distribution the better the estimate. The strength of the attack also affects the quality of the estimated value.

task. Possibly a more useful aspect of the setting in 31 is to test if a model + training method can produce a robust model. For instance in my experiments I asked:

Does a SSL model learn a more robust representation than a supervised model on this controlled dataset?

7.3 Experiments

[Küg+21] argues that discriminative learning methods such as simclr can be used to disentangle the underlying causal factors of a dataset. They create the 3dident dataset as a toy dataset to validate their theory. I initially tested if I can train the simclr model with similar performance as the paper [Küg+21].

The simclr model is trained to extract image features using self supervision. I then train a linear readout layer separately (similar to 31) and regress the causal factors. For continuous causal factors the performance of simclr is measured by R^2 (coefficient of determination). To calculate R^2 for a regression task involving n variables with actual values $\{y_1, y_2, ..., y_n\}$ and regressed values $\{f_1, f_2, ..., f_n\}$ we have:

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{23}$$

$$SS_{\text{res}} = \sum_i (y_i - f_i)^2 = \sum_i e_i^2 \tag{24}$$

$$SS_{\text{tot}} = \sum_i (y_i - \bar{y})^2 \tag{25}$$

$$R^2 = 1 - \frac{SS_{\text{res}}}{SS_{\text{tot}}} \tag{26}$$

Where residuals are defined as $e_i = y_i - f_i$, SS_{res} is the residual sum of squares and SS_{tot} is the total sum of squares. Note that SS_{tot} is only a function of the data and constant w.r.t the predictions. This means that the less residual we have for each prediction the lower SS_{res} while SS_{tot} is constant so R^2 is closer to 1. Note that we can have $SS_{tot} < SS_{res}$ which means our predictions f_is are even worse than predicting the average of y_is for all the points and this results in a negative value for R^2. I used accuracy to measure object classification performance. All the measurements were done using standard images and the results can be see in 5. For simclr I chose the small random crop and color distortion data augmentations since they provided the best R^2 score for spotlight position (rotation). As we can see the results are very close to that of the paper [Küg+21] table 1. Specifically the spotlight angle (θ) can be regressed with low error.

I also included a supervised model to compare to simclr. Similar to the CIFAR10 experiments I used lightening bolts Resnet18 as the backbone for both supervised and simclr training. Unlike CIFAR10 since the images in 3dident are much bigger, the first convolutional layer reduces the output resolution by half (112x112) this doesn't affect the results in 5 and makes training much faster. One possible choice for the continuous variable of interest is spotlight rotation

Readout layer	Object Accuracy	x	y	z	α	β	γ	θ
linear readout	1.000	0.864	0.791	0.792	0.621	0.527	0.533	0.962
kernel ridge	1.000	0.730	0.658	0.671	0.582	0.497	0.569	0.922

Table 5: The R^2 scores for different image features for Simclr. First row results for a linear readout, second row corresponds to using kernel ridge regression. Note since I used color distortion in training simclr the scores for the 3 color related attributes (object, spotlight, background) were negative or close to 0 and excluded from the table. We can see that linear readout gives the best results.

$\theta \in [-\pi/2, \pi/2]$. The spotlight position as a function of θ is:

$$(4 * sin(\theta), 4 * cos(\theta), 6 + max\_object\_size) \tag{27}$$

where "max_object_size" is the maximum dimension of the object with the biggest dimensions and is a constant. This means the spotlight moves in a half circle above the object shining at it in an angle. θ is originally uniformly sampled from $[-1, 1]$ and linearly mapped to $[-\pi/2, \pi/2]$. To create a classification task I discretized θ into 3 non-overlapping classes: $[\pi/4, \pi/2]$, $[-\pi/8, \pi/8]$, $[-\pi/2, -\pi/4]$ similar to 30. The interval lengths are equal so the classes are balanced.

I repeated the same training process as CIFAR10 to train SimCLR and Supervised models (including retraining the last layer for supervised which had the same results). Note that here, unlike the previous section, for both models, the readout layer is trained as a classifier and not a regressor. Since my readout layer for regression in 5 was a simple linear layer, from the perspective of model capacity having a linear classification readout is the same as first linearly regressing spotlight and then classifying it. The supervised model was trained only on 4% of the data since that was enough to reach an accuracy above 99% however, Simclr was trained on the whole trainset. The attacks had $\epsilon = 8/255$ with l_∞ norm. The results are:

1. Both models had standard accuracies in $[0.99, 1]$ interval.
2. All had robust accuracy 0 % for APGD with 100 iterations and all have accuracy under 1% for APGD 10 iterations.
3. The simclr model had an accuracy of 33% against FGSM, while the supervised model's accuracy was 1%. As expected this shows FGSM is much less accurate than APGD in solving the optimization problem .

In conclusion, just because contrastive methods such as simclr can retrieve the causal factors accurately it doesn't make these models any more robust than supervised models. This might be due to a well known phenomena called **underspecification**. An ML pipeline is underspecified if there are many distinct ways (e.g., different weight configurations) for the model to achieve equivalent held-out performance on iid data, even if the model specification and training data are held constant [DAm+20]. We could already observe this in the L-T dataset when training the model on only the 8 standard images. There, both of the robust and non-robust models had 100% standard accuracy, but completely opposite robust accuracies (0% and 100%). This means the problem as represented by the standard dataset was underspecified. So in order to train a robust model we need to add more conditions or constraints to the training pipeline. It might be necessary to extend the dataset with lots of data (similar to the L-T dataset extension which can be seen as a from of data augmentation) or adding more assumptions via a special model architecture, loss modification or regularization.

One major shortcoming of designing the experiment around spotlight rotation is that it's hard for humans to classify it by looking at the images while the computer does much better. This makes it much harder to intuitively reason about the shortcomings of the model, dataset or the proper amount of perturbation. So I used Blender to create a similar dataset that is based on regressing and classifying the distance between objects making it more human understandable. Future experiments can take advantage of such a dataset.

8 Conclusion

In this work I used different experiments to show that the different standard SSL training pipelines, similar to supervised training, are adversarially non-robust. And that the dataset can play an important role in model robustness.

In section 4 I examined how different loss functions and training methods affect a model's adversarial accuracy. The results show that the SSL models are not any more robust than the supervised models.

In section 6 I created a toy dataset and showed that if we include the appropriate boundary samples in the dataset we can train a robust model without using any attacks. This experimentally shows that in a learning pipeline if the dataset's classes are not ambiguous and the dataset includes all the "necessary" adversarial variations of standard samples and we have an "appropriate" model architecture capable of learning a robust model, i.e., we control the dataset and model elements of the pipeline, then the standard supervised training method available is enough to train a fully robust model [20]. Moreover, I argued that if we don't include the boundary samples in the dataset it might be impossible to train a 100% robust model.

In section 7 I created a theoretical framework to analyze the robustness of a model, training method or loss function by transforming a regression task into a classification task. I did this by separating the inherent robustness that stems from the problem definition from that of the model robustness which is the result of the training method. I further showed that just because SSL models can accurately regress the samples' causal factors, it doesn't mean these models are robust. i.e., even when controlling for the loss, dataset and model features the training method can be a major source of adversarial susceptibility via its influence on the smoothness of the model features with respect to input.

Future work could focus on creating more realistic and complex datasets similar to the extended L-T dataset in 6 and likewise try to eliminate the issue of underspecification and observe whether the results also hold for these datasets. Moreover, if a training method could efficiently control the lipschitz constant of a network it is possible to use the setup in section 7.2 to test the model's smoothness and robustness.

[20]here the meaning of adjectives in double quotes depends on the problem definition and properties.

References

[1] Jörn-Henrik Jacobsen et al. “Excessive Invariance Causes Adversarial Vulnerability”. In: (2018). DOI: 10.48550/ARXIV.1811.00401. URL: https://arxiv.org/abs/1811.00401.

[2] Christian Szegedy et al. *Intriguing properties of neural networks*. 2013. DOI: 10.48550/ARXIV.1312.6199. URL: https://arxiv.org/abs/1312.6199.

[3] Julius von Kügelgen et al. “Self-Supervised Learning with Data Augmentations Provably Isolates Content from Style”. In: *arXiv:2106.04619 [cs, stat]* (Oct. 2021). arXiv: 2106.04619. URL: http://arxiv.org/abs/2106.04619 (visited on 12/17/2021).

[4] Maksym Andriushchenko and Nicolas Flammarion. *Understanding and Improving Fast Adversarial Training*. 2020. DOI: 10.48550/ARXIV.2007.02617. URL: https://arxiv.org/abs/2007.02617.

[5] Seyed-Mohsen Moosavi-Dezfooli et al. *Robustness via curvature regularization, and vice versa*. 2018. DOI: 10.48550/ARXIV.1811.09716. URL: https://arxiv.org/abs/1811.09716.

[6] Andrew Ilyas et al. *Adversarial Examples Are Not Bugs, They Are Features*. 2019. DOI: 10.48550/ARXIV.1905.02175. URL: https://arxiv.org/abs/1905.02175.

[7] Dan Hendrycks et al. “Using Self-Supervised Learning Can Improve Model Robustness and Uncertainty”. In: *CoRR* abs/1906.12340 (2019). arXiv: 1906.12340. URL: http://arxiv.org/abs/1906.12340.

[8] Evgenii Zheltonozhskii et al. “Contrast to Divide: Self-Supervised Pre-Training for Learning with Noisy Labels”. In: *2022 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Jan. 2022. DOI: 10.1109/wacv51458.2022.00046. URL: https://doi.org/10.1109%2Fwacv51458.2022.00046.

[9] Yihao Xue, Kyle Whitecross, and Baharan Mirzasoleiman. “Investigating Why Contrastive Learning Benefits Robustness Against Label Noise”. In: *CoRR* abs/2201.12498 (2022). arXiv: 2201.12498. URL: https://arxiv.org/abs/2201.12498.

[10] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2014. DOI: 10.48550/ARXIV.1412.6572. URL: https://arxiv.org/abs/1412.6572.

[11] Francesco Croce and Matthias Hein. *Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks*. 2020. DOI: 10.48550/ARXIV.2003.01690. URL: https://arxiv.org/abs/2003.01690.

[12] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: 1706.06083 [stat.ML].

[13] David Stutz, Matthias Hein, and Bernt Schiele. “Disentangling Adversarial Robustness and Generalization”. In: *arXiv:1812.00740 [cs, stat]* (Apr. 2019). arXiv: 1812.00740. URL: http://arxiv.org/abs/1812.00740 (visited on 11/17/2021).

[14] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. "Unsupervised Representation Learning by Predicting Image Rotations". In: *arXiv:1803.07728 [cs]* (Mar. 2018). arXiv: 1803.07728. URL: http://arxiv.org/abs/1803.07728 (visited on 08/02/2021).

[15] Sanjeev Arora et al. "A Theoretical Analysis of Contrastive Unsupervised Representation Learning". In: *arXiv:1902.09229 [cs, stat]* (Feb. 2019). arXiv: 1902.09229. URL: http://arxiv.org/abs/1902.09229 (visited on 08/01/2021).

[16] Ting Chen et al. "A Simple Framework for Contrastive Learning of Visual Representations". In: *arXiv:2002.05709 [cs, stat]* (June 2020). arXiv: 2002.05709. URL: http://arxiv.org/abs/2002.05709 (visited on 09/14/2021).

[17] Jure Zbontar et al. *Barlow Twins: Self-Supervised Learning via Redundancy Reduction*. 2021. DOI: 10.48550/ARXIV.2103.03230. URL: https://arxiv.org/abs/2103.03230.

[18] Jeff Z. HaoChen et al. *Provable Guarantees for Self-Supervised Deep Learning with Spectral Contrastive Loss*. 2021. DOI: 10.48550/ARXIV.2106.04156. URL: https://arxiv.org/abs/2106.04156.

[19] Robert Geirhos et al. *ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness*. 2018. DOI: 10.48550/ARXIV.1811.12231. URL: https://arxiv.org/abs/1811.12231.

[20] Ting Chen, Calvin Luo, and Lala Li. *Intriguing Properties of Contrastive Losses*. 2020. DOI: 10.48550/ARXIV.2011.02803. URL: https://arxiv.org/abs/2011.02803.

[21] Sheng Liu et al. *Early-Learning Regularization Prevents Memorization of Noisy Labels*. 2020. DOI: 10.48550/ARXIV.2007.00151. URL: https://arxiv.org/abs/2007.00151.

[22] Joshua Robinson et al. "Can contrastive learning avoid shortcut solutions?" In: *arXiv:2106.11230 [cs]* (June 2021). arXiv: 2106.11230. URL: http://arxiv.org/abs/2106.11230 (visited on 10/12/2021).

[23] Ziyu Jiang et al. "Robust Pre-Training by Adversarial Contrastive Learning". In: *arXiv:2010.13337 [cs]* (Oct. 2020). arXiv: 2010.13337. URL: http://arxiv.org/abs/2010.13337 (visited on 11/04/2021).

[24] Ting Chen, Calvin Luo, and Lala Li. "Intriguing Properties of Contrastive Losses". In: *arXiv:2011.02803 [cs, stat]* (Oct. 2021). arXiv: 2011.02803. URL: http://arxiv.org/abs/2011.02803 (visited on 11/04/2021).

[25] Dimitris Tsipras et al. "Robustness May Be at Odds with Accuracy". In: *arXiv:1805.12152 [cs, stat]* (Sept. 2019). arXiv: 1805.12152. URL: http://arxiv.org/abs/1805.12152 (visited on 10/01/2021).

[26] William Falcon and Kyunghyun Cho. "A Framework For Contrastive Self-Supervised Learning And Designing A New Approach". In: *arXiv preprint arXiv:2009.00104* (2020).

[27] Huy Phan. *huyvnphan/PyTorch_CIFAR10*. Version v3.0.1. Jan. 2021. DOI: 10.5281/zenodo.4431043. URL: https://doi.org/10.5281/zenodo.4431043.

[28] Xinlei Chen and Kaiming He. *Exploring Simple Siamese Representation Learning*. 2020. DOI: 10.48550/ARXIV.2011.10566. URL: https://arxiv.org/abs/2011.10566.

[29] Yang You, Igor Gitman, and Boris Ginsburg. *Large Batch Training of Convolutional Networks*. 2017. DOI: 10.48550/ARXIV.1708.03888. URL: https://arxiv.org/abs/1708.03888.

[30] Ting Chen, Calvin Luo, and Lala Li. "Intriguing Properties of Contrastive Losses". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 11834–11845. URL: https://proceedings.neurips.cc/paper/2021/file/628f16b29939d1b060af49f66ae0f7f8-Paper.pdf.

[31] Roland S. Zimmermann et al. *Contrastive Learning Inverts the Data Generating Process*. 2021. DOI: 10.48550/ARXIV.2102.08850. URL: https://arxiv.org/abs/2102.08850.

[32] Saeed Mahloujifar et al. "Empirically Measuring Concentration: Fundamental Limits on Intrinsic Robustness". In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper/2019/file/46f76a4bda9a9579eab38a8f6eabcda1-Paper.pdf.

[33] Alexander D'Amour et al. *Underspecification Presents Challenges for Credibility in Modern Machine Learning*. 2020. DOI: 10.48550/ARXIV.2011.03395. URL: https://arxiv.org/abs/2011.03395.