When does visual self-supervision aid adversarial training in improving adversarial robustness?

Michal Kucer
Los Alamos National Lab
michal@lanl.gov

Diane Oyen
Los Alamos National Lab
doyen@lanl.gov

Garrett Kenyon
Los Alamos National Lab
gkenyon@lanl.gov

Abstract

Recent self-supervision methods have found success in learning feature representations that could rival ones from full supervision, and have been shown to be beneficial to the model in several ways: for example improving model robustness and out-of-distribution detection. We conduct an empirical study to answer the question: when does visual self-supervision (SSL), as a pretext task, aid adversarial training in improving robustness to adversarial perturbations? Including visual self-supervision as part of adversarial training can indeed improve model robustness, however it turns out the devil is in the details. To improve model robustness for large values of adversarial perturbation, one should take the approach of adversarial semi-supervised ensemble training (ASSET) to generate images that are trying to fool both the self-supervised and supervised training objectives.

1 Introduction

Though neural networks achieve state of the art results in many tasks and domains, much recent work has focused on defending against and understanding adversarial examples, one of the main weaknesses of deep models [30, 11, 35]. Adversarial examples are inputs which have been imperceptibly changed and cause a model to provide erroneous output. However, recent work suggests that leveraging *self-supervised learning* can lead to improvements in the robustness of models [4, 9, 15, 29]. Self-supervised learning [10, 25, 3] and unsupervised learning [8, 18] were developed to learn generalizable representations without requiring labeled data. Self-supervised learning (SSL) leverages the structure of the data to learn representations through optimizing the network weights via an auxiliary task: e.g. in-painting [28], rotation prediction [10], or image colorization [38]. The goal of this paper is to understand how incorporating self-supervision into adversarial training [24] affects model robustness to adversarial perturbations and common image corruptions [14], as there is more to using self-supervision to improve robustness than meets the eye.

Contributions We find that the best way to leverage self-supervision for boosting model robustness is via adversarial semi-supervised ensemble training (ASSET), however it could penalize the non-robust accuracy. This work presents a comprehensive study to understand various ways one can combine self-supervision with adversarial training and observe the effects on adversarial robustness against adversarial perturbations.

Related Work Our work is most closely related to [4, 9, 15–17, 29], which explore using self-supervision to improve model robustness. [4, 15] demonstrate that self-supervision improves model robustness via self-supervised adversarial pre-training (SSAPT) and as an auxiliary loss. [9, 16, 17] propose methods to learn robust representations via self-supervised contrastive learning methods without labeled data. Sun et al. [29] study the use of SSAPT and using pre-text task loss as a form of auxiliary supervision (akin to $\mathcal{T}_1$, described in Sec. 2) in the setting of robust 3D point cloud

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

recognition. Our empirical study significantly extends our understanding of the effect self-supervision can have on model robustness.

Note that our work is complimentary to self-supervised pre-training approaches [4, 9, 16, 17]. Our methods can be pipelined with the various pre-training approaches to potentially further enhance model robustness. In our experiments, we study the following self-supervised tasks: rotation prediction [10], and solving a jigsaw puzzle [2, 25]. Both of these tasks are implemented as a supervised task with multi-class cross-entropy loss as a criterion to be optimized.

2 Improving robustness with self-supervision

In this section, we describe the various approaches to combine self-supervised learning (SSL) with adversarial training (AT). Please see Section A.3, which provides background on adversarial training and generation of adversarial examples. The baseline AT method of projected gradient descent (PGD) attack [24] is denoted as $\mathcal{T}_0$. One can use image rotations (or other self-supervision) in the image pre-processing pipeline for data augmentation (without incorporating self-supervised loss).

Adversarial training with self-supervision The first way to incorporate SSL into AT is to apply the self-supervised transformation T to the batch of images X to obtain transformed versions of the batch and corresponding labels X_{t} and y_{t} respectively. One then computes the adversarial version of $X_{\mathrm{t}}^{\mathrm{adv}}$ and minimizes the following loss to optimize model weights:

$$\mathcal{L}_{\mathcal{T}_{\{1,2\}}} = \mathcal{L}_{\text{sup}}(F_\theta(X_{\mathrm{t}}^{\mathrm{adv}}), y) + \lambda \cdot \mathcal{L}_{\mathrm{t}}(F_\theta^{\mathrm{t}}(X_{\mathrm{t}}^{\mathrm{adv}}), y_{\mathrm{t}}), \tag{1}$$

where λ is the weight of the self-supervised loss , F_θ and F_θ^{t} together represent a neural network model that takes as input a batch $X_{\mathrm{t}}^{\mathrm{adv}}$ and two prediction heads, one for the supervised task $(F_\theta(\cdot), y)$ and the other for the self-supervised task $(F_\theta^{\mathrm{t}}(\cdot), y_{\mathrm{t}})$ with corresponding loss functions $\mathcal{L}_{\text{sup}}$ and $\mathcal{L}_{\mathrm{t}}$. This approach is denoted either $\mathcal{T}_1$ or $\mathcal{T}_2$ depending on the next model choice. We have to make a choice whether to use the self-supervision loss $\mathcal{L}_{\mathrm{t}}$ in the PGD attack, which results in the following expression for the loss $\mathcal{L}_{\text{PGD}}$ used to generate adversarial examples (see Section A.3 for detailed background on using PGD to generate adversarial examples):

$$\mathcal{L}_{\text{PGD}} = \mathcal{L}_{\text{sup}}(F_\theta(X^k), y) + w_1 \cdot \mathcal{L}_{\mathrm{t}}(F_\theta^{\mathrm{t}}(X_{\mathrm{t}}^k), y_{\mathrm{t}}). \tag{2}$$

When we use just $\mathcal{L}_{\text{sup}}$ to generate adversarial images ($w_1 = 0$ in Equation 2), we denote this variation as $\mathcal{T}_1$. If $w_1 > 0$, we denote this variation as $\mathcal{T}_2$. Note that in variation $\mathcal{T}_2$ we are essentially crafting an adversarial example against an ensemble of known classifiers (supervised and self-supervised classifiers). $\mathcal{T}_2$ can therefore be viewed as a variation of the average PGD attack [23], and a particular variation of the "Ensemble Attack over Multiple Models" [34], in which we assign the importances 1 and w to the supervised and self-supervised models respectively.

Augmenting training with self-supervised examples Another way to add SSL to AT is by augmenting the training set with self-supervised examples by making a copy of the batch X to which the self-supervised transformation T will be applied, resulting in a batch (X, y) and a batch $(X_{\mathrm{t}}, y_{\mathrm{t}}) = T(X)$. The simplest way to add self-supervision into AT with a separate batch (denoted as $\mathcal{T}_3$) is by generating adversarial examples X^{adv}, while leaving X_{t} unchanged, and minimizing the following joint loss,

$$\mathcal{L}_{\mathcal{T}_3} = \mathcal{L}_{\text{sup}}(F_\theta(X^{\mathrm{adv}}), y) + \lambda \cdot \mathcal{L}_{\mathrm{t}}(F_\theta^{\mathrm{t}}(X_{\mathrm{t}}), y_{\mathrm{t}}). \tag{3}$$

Lastly, one can attack both the original batch images X and transformed images X_{t}, resulting in the following loss that will be used to optimize the network parameters.

$$\mathcal{L}_{\mathcal{T}_{\{\cdot\}}} = \mathcal{L}_{\text{sup}}(F_\theta(X^{\mathrm{adv}}), y) + \lambda \cdot \mathcal{L}_{\mathrm{t}}(F_\theta^{\mathrm{t}}(X_{\mathrm{t}}^{\mathrm{adv}}), y_{\mathrm{t}}). \tag{4}$$

When generating adversarial examples for X and X_{t}, should one use the supervised loss, the self-supervised loss, or both to find the worst case adversarial examples? As X and X_{t} are both passed into the PGD function together (stacked), this results in the following expression for $\mathcal{L}_{\text{PGD}}$:

$$\mathcal{L}_{\text{PGD}} = \mathcal{L}_{\text{sup}}(F_\theta(X^k), y) + w_1 \cdot \mathcal{L}_{\mathrm{t}}(F_\theta^{\mathrm{t}}(X_{\mathrm{t}}^k), y_{\mathrm{t}}) + w_2 \cdot \mathcal{L}_{\text{sup}}(F_\theta(X_{\mathrm{t}}^k), y) + w_3 \cdot \mathcal{L}_{\mathrm{t}}(F_\theta^{\mathrm{t}}(X^k), \mathbf{0}). \tag{5}$$

For majority of the experiments, we set w_2 and w_3 to 0 in order to simplify ablation experiments. We consider the following two methods $\mathcal{T}_4$ and $\mathcal{T}_5$. They both minimize Eq. 4, however the difference between $\mathcal{T}_4$ and $\mathcal{T}_5$ is that for $\mathcal{T}_4$, w_1 is set to 0, whereas it is positive for $\mathcal{T}_5$. Incorporating the self-supervision task loss into PGD can be viewed as a form ensemble adversarial training [31], where we augment the training set with adversarial examples crafted from the self-supervised task, and thus calling it *adversarial semi-supervised ensemble training* (ASSET).

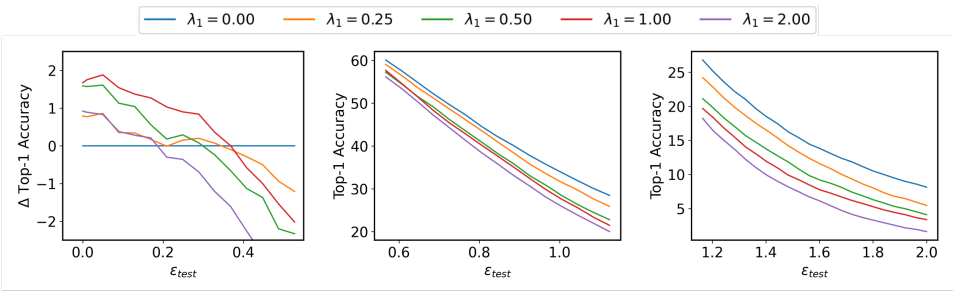

Figure 1: Comparison of models trained with $\mathcal{T}_3$ for which the weight of the self-supervised loss, λ, is varied. The plots highlight different parts of the domain to better illustrate the trend. Leftmost plot rather than showing the performance of the various models, shows the comparison of performance for the given model, e.g. the line in left plot that shares the yellow colour with the plot in the middle or right plot, shows the difference in performance $\text{Acc}(\lambda = 0.25) - \text{Acc}(\lambda = 0)$.

3 When does including SSL into AT improve model robustness?

In this section, we compare the ways SSL can be incorporated into AT against the baseline AT procedure of Madry et al. [24] ($\mathcal{T}_0$). Table 1 summarizes the general trends we saw for l_∞-robust performance of the various methods for the ResNet-18 architecture on CIFAR-10 dataset.

Our first set of experiments explored the scenario in which we apply the self-supervised transformation $T(\cdot)$ to an image batch X, to obtain X_t. In the first approach, $T(\cdot)$ is used as part of the pre-processing pipeline, however the SSL labels y_t are not used to optimize the model. Though image rotations are often included in augmentation pipelines, they rotate the image through a much smaller range of angles (e.g. pick a random rotation between $-45°$ to $45°$). As we can see from the second line of Table 1, applying extreme rotations significantly degrades adversarial training performance, dropping the test accuracy (TA) and robust accuracy at $\epsilon_\text{test} = 8/255$ ($\text{RA}_{8/255}$) from 85.09% and 46.06% to 72.44% and 40.88% respectively. Using $\mathcal{L}_\text{t}$ as an auxiliary loss ($\mathcal{T}_1$ and $\mathcal{T}_2$) helps regularize learning and improves the TA and RA for several values of ϵ_test as compared to "$\mathcal{T}_0$ w/ rot aug.". From these results, we can see that replacing the supervised task with self-supervised tasks which significantly modify or distort the data are likely to harm adversarial training. As we will show, a better way improve robustness is to augment the training set with self-supervised examples.

In the following set of experiments we (1) augment X with X_t (unlike $\mathcal{T}_0$, $\mathcal{T}_1$, $\mathcal{T}_2$) and (2) vary how X_t is modified during adversarial training with the variations described by $\mathcal{T}_3$, $\mathcal{T}_4$, and $\mathcal{T}_5$. In the training scenario $\mathcal{T}_3$, we optimize the empirical loss from Equation 3 , where $\mathcal{L}_\text{sup}$ is the supervised portion of the loss operating on PGD-attacked batch X^adv, and $\mathcal{L}_\text{t}$ is the self-supervised loss (SSL) operating on clean batch X_t. Figure 1 illustrates general trends we see for $\mathcal{T}_3$, as the self-supervision importance (λ) and the level of robustness tested (ϵ_test) vary. The tabulated results for various values of ϵ_test and λ can be found in the Appendix Table C1. In general, we see that TA is higher for models with $\lambda > 0$. The use of self-supervision ($\lambda > 0$) consistently improves the TA and RA for several values of $\epsilon_\text{test} < \epsilon_\text{train}$, as compared to baseline $\mathcal{T}_0$ (see left plot of Figure 1). Furthermore, larger values of λ result in larger increase in TA and RA for several values of $\epsilon_\text{test} < \epsilon_\text{train}$. However, for $\epsilon_\text{test} > \epsilon_\text{train}$ (middle and right plots of Figure 1) we see the opposite behavior — RA drops further as compared to the baseline the larger the weight λ. From Figure 1, and Table 1, we can see that $\mathcal{T}_3$ is very effective in regularizing the model training and provides the largest increase in TA as compared to other methods (though sacrificing RA). The rows $\mathcal{T}_4$ and $\mathcal{T}_5$ from Table 1 show the cases in which both original images (X) and transformed images (X_t) are passed into PGD and use the loss from Equation 4 for network parameter optimization. For the case of $\mathcal{T}_4$ we have $w_1 = 0$ and we see that the TA and RA are better than $\mathcal{T}_0$ for all values of ϵ_test, though TA is worse than $\mathcal{T}_3$. Lastly, for $\mathcal{T}_5$ we have $w_1 > 0$ and we see that TA in general either does not change or gets worse compared to all the other methods, showing that if one wants to optimize the model for clean accuracy, one should not use SSL loss in generating adversarial examples. However, $\mathcal{T}_5$ shows the largest increase in RA as compared to the baseline and the other methods, especially for larger values of ϵ_test. What causes this drop in TA and increase in RA? By having two separate batches of images (X and X_t), and attacking both the supervision and self-supervision loss, we are performing a variation of ensemble adversarial training (EAT) [31], in which we augment the training set with adversarial examples from the self-supervised classifier. As can be seen in the last paragraph of Section 2, there are two

Table 1: Mean Top-1 accuracy (averaged over 10 trials) on the CIFAR-10 test set with ResNet-18 models trained with l_∞ norm and $\epsilon_{\text{train}} = 8/255$ evaluated at different values of ϵ_{test} used to generate the robust test set as we vary the training method and SSL involvement.

Method / ϵ_{test}	$0/255$	$3/255$	$4/255$	$5/255$	$6/255$	$7/255$	$8/255$	$9/255$	$10/255$
$\mathcal{T}_0$ [24]	85.16	72.73	67.76	62.55	57.06	51.47	46.00	40.82	35.83
$\mathcal{T}_0$ w/ rot aug.	72.54	61.90	57.82	53.70	49.38	45.07	40.68	36.38	32.19
$\mathcal{T}_1$	74.60	63.60	59.41	55.04	50.57	45.96	41.34	36.82	32.47
$\mathcal{T}_2$	74.51	63.10	58.67	54.27	49.67	45.16	40.63	36.05	31.79
$\mathcal{T}_3$	**86.49**	73.16	67.64	61.64	55.51	49.54	43.59	38.17	33.03
$\mathcal{T}_4$	85.90	**73.95**	**68.82**	**63.27**	57.65	51.85	46.23	40.77	35.76
$\mathcal{T}_5$	84.54	72.70	67.97	62.89	57.60	52.23	46.88	**41.67**	**36.80**
ASSET (Best)	85.15	73.19	68.35	**63.27**	**57.84**	**52.50**	**46.94**	**41.66**	**36.82**

additional terms $w_2 \cdot \mathcal{L}_{\text{sup}}(F_\theta(X_{\text{t}}^k), y)$ and $w_3 \cdot \mathcal{L}_{\text{t}}(F_\theta^{\text{t}}(X^k), \mathbf{0})$ that can be added to $\mathcal{L}_{\text{PGD}}$: w_2 term adds SSL perturbations to the original batch, w_3 adds original perturbations to the SSL batch. We perform evaluation by varying w_2 and w_3 while keeping w_1 set to 0.5 as this setting resulted in the highest gain in RA. We found that the best setting for our parameters is $w_1 = 0.5$, $w_2 = 0.0$, $w_3 = 0.5$, which is shows in the last row of Table 1. By further adding adversarial perturbation from the supervised task to $X_{\text{t}}^{\text{adv}}$, we are able to preserve the TA, while boosting the RA.

Discussion. From our empirical study, we see that for self-supervised tasks that make a significant modification to the original data batch, it is crucial to generate a separate batch on which the SSL task is applied (augmenting rather than replacing the original data). In general, leveraging the SSL task loss as an auxiliary loss is helpful in regularizing AT and improving TA and RA for several values of ϵ_{test}. Furthermore, we can see that both (a) combining the supervision with self-supervised loss, and (b) attacking both the supervision and self-supervision losses in PGD play a vital role in affecting the TA and robustness to adversarial perturbations, which is contrary to findings of [4]. Just adding SSL loss ($\mathcal{T}_3$ with $\lambda > 0$) improves the TA and RA for $\epsilon_{\text{test}} \leq \epsilon_{\text{train}}$, however it hurts RA for larger values of ϵ_{test}. Using the adversarial version of both X and X_{t} using only the supervised loss ($\mathcal{T}_4$) slightly improves the accuracy for all ϵ_{test} as compared to $\mathcal{T}_3$, however slightly lowers TA and RA for small values of ϵ_{test}. $\mathcal{T}_5$ can be viewed as a form of a ensemble adversarial training (EAT) [31], resulting in an increase in model robustness to larger perturbations however sacrificing TA.

How beneficial is pre-training? Another question is: how beneficial is SSAPT to our most robust model, $\mathcal{T}_5$. Please see Table C2 for a comparison of the training methods with the different pre-training scenarios. Similar to [4], we see a boost in model robustness when SSAPT is followed by AT ($\mathcal{T}_0$). However, we do not see any significant benefit from SSAPT if AT includes self-supervision during training ($\mathcal{T}_5$).

4 Conclusion

We provide a comprehensive study on the effects of SSL and training choices on model robustness, with explanations of observed trends. This work is a solid platform for exploring and understanding the ways that incorporating SSL into AT affects model accuracy and robustness; with findings that reach significantly beyond results from other published work. As we saw, the task of incorporating SSL into AT is not straightforward and requires care. Augmenting the training set by images transformed by the self-supervised task and incorporating self-supervised task loss into the training objective provide a significant boost in standard testing accuracy, though can hurt the robust testing accuracy for larger values of ϵ_{test}. Instead, if the main goal is to improve model robustness for large values of ϵ_{test}, one should take the approach of adversarial semi-supervised ensemble training (ASSET) and generate images that are trying to fool both the self-supervised and supervised training objectives. Though ASSET could result in a drop in clean accuracy, we see gains in robustness against adversarial perturbations and natural image corruptions (Sec. C.2). Additionally, we show that these benefits can be had with different SSL tasks (Sec. C.1), on various datasets and using various base architectures (Sec. C.3).

References

[1] Anish Athalye, Nicholas Carlini, and David A. Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *ICML*, pages 274–283, 2018. URL http://proceedings.mlr.press/v80/athalye18a.html.

[2] Fabio Maria Carlucci, Antonio D'Innocente, Silvia Bucci, Barbara Caputo, and Tatiana Tommasi. Domain generalization by solving jigsaw puzzles. In *CVPR*, 2019.

[3] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments. 2020.

[4] T. Chen, S. Liu, S. Chang, Y. Cheng, L. Amini, and Z. Wang. Adversarial robustness: From self-supervised pre-training to fine-tuning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 696–705, 2020. doi: 10.1109/CVPR42600.2020.00078. URL https://doi.ieeecomputersociety.org/10.1109/CVPR42600.2020.00078.

[5] Xinlei Chen, Haoqi Fan, Ross B. Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *CoRR*, abs/2003.04297, 2020. URL https://arxiv.org/abs/2003.04297.

[6] Guneet S. Dhillon, Kamyar Azizzadenesheli, Zachary C. Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. *CoRR*, abs/1803.01442, 2018. URL http://arxiv.org/abs/1803.01442.

[7] Logan Engstrom, Andrew Ilyas, Hadi Salman, Shibani Santurkar, and Dimitris Tsipras. Robustness (python library), 2019. URL https://github.com/MadryLab/robustness.

[8] Dumitru Erhan, Yoshua Bengio, Aaron Courville, Pierre-Antoine Manzagol, Pascal Vincent, and Samy Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(19):625–660, 2010. URL http://jmlr.org/papers/v11/erhan10a.html.

[9] Lijie Fan, Sijia Liu, Pin-Yu Chen, Gaoyuan Zhang, and Chuang Gan. When does contrastive learning preserve adversarial robustness from pretraining to finetuning? In A. Beygelzimer, Y. Dauphin, P. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, 2021. URL https://openreview.net/forum?id=70kOIgjKhbA.

[10] Spyros Gidaris, Praveer Singh, and Nikos Komodakis. Unsupervised representation learning by predicting image rotations. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=S1v4N2l0-.

[11] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015. URL http://arxiv.org/abs/1412.6572.

[12] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples, 2015.

[13] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SyJ7ClWCb.

[14] Dan Hendrycks and Thomas Dietterich. Benchmarking neural network robustness to common corruptions and perturbations. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=HJz6tiCqYm.

[15] Dan Hendrycks, Mantas Mazeika, Saurav Kadavath, and Dawn Song. Using self-supervised learning can improve model robustness and uncertainty. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/a2b15837edac15df90721968986f7f8e-Paper.pdf.

[16] Ziyu Jiang, Tianlong Chen, Ting Chen, and Zhangyang Wang. Robust pre-training by adversarial contrastive learning. 2020.

[17] Minseon Kim, Jihoon Tack, and Sung Ju Hwang. Adversarial self-supervised contrastive learning. In *Advances in Neural Information Processing Systems*, 2020.

[18] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[19] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

[20] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[21] Alexey Kurakin, Ian Goodfellow, Samy Bengio, et al. Adversarial examples in the physical world, 2016.

[22] Yan LeCun and Ishan Mishra. Self-supervised learning: The dark matter of intelligence, 2021. Accessed 2021-03-04.

[23] Jiayang Liu, Weiming Zhang, and Nenghai Yu. CAAD 2018: Iterative ensemble adversarial attack. *CoRR*, abs/1811.03456, 2018. URL `http://arxiv.org/abs/1811.03456`.

[24] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=rJzIBfZAb`.

[25] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *ECCV*, pages 69–84. Springer International Publishing, 2016. ISBN 978-3-319-46466-4.

[26] Tianyu Pang, Kun Xu, Chao Du, Ning Chen, and Jun Zhu. Improving adversarial robustness via promoting ensemble diversity. In *International Conference on Machine Learning*, pages 4970–4979. PMLR, 2019.

[27] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[28] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context encoders: Feature learning by inpainting. In *Computer Vision and Pattern Recognition (CVPR)*, 2016.

[29] Jiachen Sun, Yulong Cao, Christopher B Choy, Zhiding Yu, Anima Anandkumar, Zhuoqing Morley Mao, and Chaowei Xiao. Adversarially robust 3d point cloud recognition using self-supervisions. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 15498–15512. Curran Associates, Inc., 2021. URL `https://proceedings.neurips.cc/paper/2021/file/82cadb0649a3af4968404c9f6031b233-Paper.pdf`.

[30] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[31] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv*, 2020.

[32] Trieu H. Trinh, Minh-Thang Luong, and Quoc V. Le. Selfie: Self-supervised pretraining for image embedding. *CoRR*, abs/1906.02940, 2019. URL `http://arxiv.org/abs/1906.02940`.

[33] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=SyxAb30cY7`.

[34] Jingkang Wang, Tianyun Zhang, Sijia Liu, Pin-Yu Chen, Jiacen Xu, Makan Fardad, and Bo Li. Adversarial attack generation empowered by min-max optimization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[35] Eric Wong and Zico Kolter. Provable defenses against adversarial examples via the convex outer adversarial polytope. In *International Conference on Machine Learning*, pages 5286–5295. PMLR, 2018.

[36] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations*, 2018. URL `https://openreview.net/forum?id=Sk9yuql0Z`.

[37] Dejing Xu, Jun Xiao, Zhou Zhao, Jian Shao, Di Xie, and Yueting Zhuang. Self-supervised spatiotemporal learning via video clip order prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[38] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *ECCV*, 2016.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See the last paragraph of Section 3.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Instructions are provided. We will provide code after internal review for release.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Table C4 contains the mean and standard deviation for the main ablation experiments presented in Table 1 .
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Section B contains details on the amount of compute and types of resources used.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Appendix: Preliminaries

A.1 Self-supervised learning

Self-supervised learning has gained popularity in recent years as a method of learning feature representations without the need for labeled data in domains such as images [10] and video [37]. Many visual self-supervised learning methods use the particular structure of the data to modify the samples and generate pseudo-labels that can then be fit using supervised learning, e.g. image colorization [38], or rotation prediction [10]. [25] learn features by solving jigsaw puzzles on natural images. [38] learn image features by learning to predict a plausible color version of a photograph from its grayscale version. [10] learn visual features by predicting the nature of a geometrical transformation applied to the image, in this case image rotation. [32] learn visual features by learning to predict which image patches correctly fill in a masked out image among a set of "distractor" patches. Further, some of the recent methods such as SwAV [3] or MoCo-V2 [5] utilize contrastive learning and feature clustering to learn strong feature representations.

As [22] points out, "self-supervised learning obtains supervisory signals from the data itself". We are given an input image x and we apply to it a transformation $T(\cdot)$ that gives us the a transformed version of the image x_{t} and label y_{t}:

$$(x_{\mathrm{t}}, y_{\mathrm{t}}) = T(x). \tag{6}$$

In our experiments, we study the following self-supervised tasks: rotation prediction [10], and solving a jigsaw puzzle [25, 2] (see Figure A1). Both of these tasks are implemented as a supervised task with multi-class cross-entropy loss as a criterion to be optimized. This allows us to easily incorporate rotation (or jigsaw) prediction as an auxiliary task to be optimized in addition to classification. In the case of rotation prediction, the transformation $T(\cdot)$ consists of rotating the image by a multiple of 90°, and returning a rotated image x_{t} and a label y_{t} designating the degree of rotation. In the case of the jigsaw puzzle, the task consists of dividing the image into an $n \times n$ grid, and randomly picking a set of J permutations (each assigned its own label). The transformation $T(\cdot)$ then consists of scrambling the image parts according to the randomly chosen permutation, returning the scrambled image and label associated with the permutation. The task of the network is then to either predict the rotation of the image, or the permutation that would unscramble the image.

A.2 Adversarial robustness

Though neural networks have been in the forefront of state-of-the-art research in many branches of computer vision, they exhibit some weaknesses, among them vulnerability to adversarial examples [30, 12], which completely change the output of the model. Recent years have seen many attempts to propose defenses against such examples. [6] proposes stochastic activation pruning, a strategy which randomly prunes a subset of activations. [13] uses input transformation (e.g. JPEG compression) to defend against adversarial attacks. [36] attempts to defend against adversarial attacks by adding an additional layer that randomly rescales the input. However, as [1] shows, many of these defenses can be easily circumvented. One of the most effective approaches to defending against adversarial attacks is shown to be adversarial training [24, 1].

A.3 Defending against adversarial perturbations

To set the setting, we follow the notation from [24, 33]. Given a distribution of data, $\mathcal{D}$, from which pairs (x, y) of features $x \in \mathbb{R}^d$ and labels $y \in [q]$ are sampled; a machine learning model F, parameterized by θ (denoted F_θ), is typically optimized to minimize the expected loss $\mathcal{L}$:

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\mathcal{L}(x, y; F_\theta)\right]. \tag{7}$$

To improve adversarial robustness, the goal of adversarial training is to train models that minimize the adversarial loss,

$$\mathop{\mathbb{E}}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta\in\Delta} \mathcal{L}(x+\delta, y; F_\theta)\right], \tag{8}$$

where Δ is a set of l_p-bounded perturbations, or alternatively a set S of adversarial examples $x' \in \mathcal{X}$ where $S = \{x' \in \mathcal{X} : \|x' - x\|_p < \epsilon\}$ for some perturbation strength ϵ.

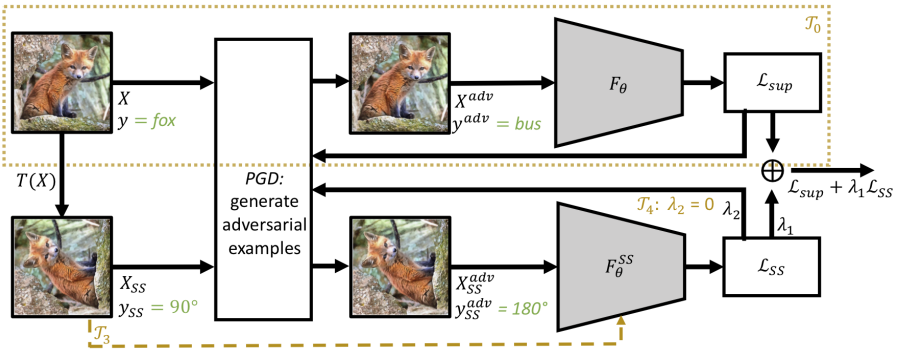

Figure A1: Overview of the ASSET model for adversarial training with self-supervision. The top half of the diagram, denoted $\mathcal{T}_0$, is a baseline adversarial training approach. As we will show, the most successful way to include SSL into AT is by way of augmenting the training set with a copy of the data batch to which the self-supervised transformation is applied. Then self-supervision can be added as an additional loss function in three ways: $\mathcal{T}_3$ includes a self-supervision loss but no adversarial attacks on self-supervised images; $\mathcal{T}_4$ adversarial attacks only consider the supervised loss; and $\mathcal{T}_5$ (the full diagram) is our approach combining the supervised and self-supervised losses in adversarial attacks. Note that $\mathcal{T}_5$ is a form of adversarial semi-supervised ensemble training (ASSET), in which we use adversarial examples for a supervised model and self-supervised model. A similar diagram for $\mathcal{T}_1$ and $\mathcal{T}_2$ is in the Supplement.

Algorithm 1 Finding adversarial examples

input: Robustness parameter ϵ, attack learning rate α, number of attack steps K, image batches X and X_{t}, labels y and y_{t}, self-supervision weight w_1
output: Adversarial images X and optionally X_{t}
Randomly perturb images X (and optionally X_{t})
$X^0 = X + U(-\epsilon, \epsilon)$, $X_{\mathrm{t}}^0 = X_{\mathrm{t}} + U(-\epsilon, \epsilon)$
for $k = 0, \ldots, K-1$ **do**
  $\mathcal{L}_{\mathrm{PGD}} = \mathcal{L}_{\mathrm{sup}}(F_\theta(X^k), y)$
  if should use self-supervised loss **then**
    $\mathcal{L}_{\mathrm{PGD}} \mathrel{+}= w_1 \cdot \mathcal{L}_{\mathrm{t}}(F_\theta^{\mathrm{t}}(X_{\mathrm{t}}^k), y_{\mathrm{t}})$
  end if
  $X^{k+1} = \Pi_S \left(X^k + \alpha \cdot \mathrm{sign}\left(\nabla_x \mathcal{L}_{\mathrm{PGD}}\right)\right)$
end for

One of the most successful methods for defending is adversarial training [11, 24, 35]. A variant of adversarial training proposed by [24] solves the above problem by finding the worst case l_p bounded adversarial examples using projected gradient descent (PGD) and finds set of parameters θ that minimize the empirical training loss using these examples [1],

$$\theta^{=} \arg\min_\theta \mathop{\mathbb{E}}_{(x,y)\in X} \left[\max_{\delta \in [-\epsilon, \epsilon]} \mathcal{L}(x + \delta, y; F_\theta) \right]. \tag{9}$$

In adversarial training by [24], an image x is first perturbed in its ϵ-neighborhood, i.e.

$$x_0 = x + U(-\epsilon, \epsilon), \tag{10}$$

where U is a uniform distribution. It then follows the generation process of the Basic Iterative Method [21, 26],

$$x^{k+1} = \Pi_S \left(x_k + \alpha \cdot \mathrm{sign}\left(\nabla_x \mathcal{L}_{\mathrm{PGD}}(x + \delta, y; F_\theta)\right)\right) \tag{11}$$

where K is the number of "attacks steps", or iterations of projected gradient descent performed to find the worst case l_p bounded adversarial example x^{adv}. Π_p is an operator that will project the iterates onto an l_p ball, where in our case $p = 2$ or ∞. Algorithm 1 describes the general procedure for generating adversarial examples using PGD attacks.

A.4 Self-supervised pre-training

From [4] we know that self-supervised adversarial pre-training (SSAPT) can help robustness. Therefore, we evaluate whether SSAPT further aids our model robustness. We consider the following pre-training settings for initializing our weights: random initialization (baseline), SSAPT with l_2 perturbations (which we call $\mathcal{P}_1$), and SSAPT with l_∞ (which we call $\mathcal{P}_2$). For pre-training, SSAPT uses X_t and y_t in the AT algorithm of [24].

B Implementation details

All of our code is implemented in Python using the PyTorch [27] deep learning framework. In pursuit of simplicity and reproducibility, our code extends the *robustness* library [7][1] (released under the MIT License). Our code is developed on an internal cluster, where each server node is equipped with 4 NVIDIA Tesla A100 cards (each with 40 GB of VRAM), paired with a dual 64-core AMD Epyc CPU and 256GB of memory. Each ablation experiment ran on average for 4 hours on a single node. Our ablation experiments utilize the ResNet-18 architecture[2].

When applying self-supervised transformation $T(\cdot)$ to X, each image is randomly rotated by ϕ degrees, where $\phi \in \{0°, 90°, 180°, 270°\}$. This implementation differs from [15], as they simultaneously generate and predict all four possible rotations for each image in the batch (if the size of X is b samples, X_t is of size $4 \cdot b$). Furthermore, when leveraging SS loss to guide adversarial training (step 6 of Algorithm 1), we use the following form of the loss:

$$\mathcal{L}_\mathrm{t}(X_\mathrm{t}, y_\mathrm{t}) = \frac{1}{N} \sum_{i \in \{1,\cdots,n\}} \mathcal{L}_{CE}(F_\theta^\mathrm{t}(x_\mathrm{t}^i), y_\mathrm{t}^i), \tag{12}$$

which differs from [15] in that they omit the averaging term $1/N$. In our experiments, averaging reduction for the loss gives consistently better results.

B.1 Datasets

To select our best models, ablation experiments are conducted on CIFAR-10 [19]. We validate our best models on CIFAR-10 and CIFAR-100. The CIFAR-10C dataset [14], used to test our model against common corruptions, is release under the Apache 2.0 License[3]. Note that CIFAR10[20] and CIFAR100[20] dataset do not have license attached to them. The CIFAR10[20] and CIFAR100[20] are publicly available on Alex Krizhevsky's website[4], and are a subset of the TinyImages dataset, that is no longer available[5].

B.2 Training and evaluation details

All of the ablation experiments investigate the addition of SSL into AT are trained for 100 epochs, with a starting learning rate of 0.1 (reduced by 10 every 40 epochs) and optimized using Stochastic Gradient Descent (SGD) with momentum 0.9. The CIFAR-10 training set is further split into a training and validation set (15% or 7500 images are used for validation and the rest are used for training). For adversarial training and evaluation, we use 10 and 20-step l_p PGD attacks [24]. For l_∞ adversaries, $\epsilon_\text{train} = 8/255$ and the attack learning rate is $\alpha = 2/255$ [15, 4]. Similar to [4], in our results we refer to the non-robust, i.e. $\epsilon = 0$, accuracy as the *Standard Testing Accuracy* (TA), and robust accuracy at $\epsilon_\text{test} = \epsilon$ as the *Robust Testing Accuracy* (RA_ϵ). For a particular set of hyper-parameters, we pick the model with the highest validation TA and test across a range of ϵ_test values. The adversarial perturbations for testing images are only generated using the supervision loss in the PGD attack.

[1] Source code can be made available upon internal review.

[2] Note that the robustness package uses slightly different architectures for the CIFAR-10 and ImageNet dataset. CIFAR-10 uses slightly smaller convolutional kernels. For more details, please see: https://github.com/MadryLab/robustness

[3] Publicly available at `https://github.com/hendrycks/robustness`

[4] `https://www.cs.toronto.edu/~kriz/cifar.html`

[5] Please see `http://groups.csail.mit.edu/vision/TinyImages/`

Table C1: Top-1 accuracy on the CIFAR-10 validation set with ResNet-18 models trained with l_∞ norm and $\epsilon_{train} = 8/255$ evaluated at different values of ϵ_{test} used to generate the robust validation set as we vary λ.

Method / ϵ_{test}	$0/255$	$3/255$	$4/255$	$5/255$	$6/255$	$7/255$	$8/255$	$9/255$	$10/255$
$\lambda = 0.00$	85.46	72.09	66.98	61.29	55.83	50.19	44.69	39.38	34.60
$\lambda = 0.25$	86.31	73.62	68.17	62.56	56.52	50.19	44.33	39.04	33.98
$\lambda = 0.50$	86.59	73.69	68.71	62.86	56.88	50.52	44.51	38.90	33.89
$\lambda = 1.00$	85.04	71.34	65.80	59.72	53.60	47.48	41.59	36.27	31.36
$\lambda = 2.00$	82.03	68.09	62.77	57.07	50.65	44.46	38.70	33.12	27.96

Table C2: Top-1 accuracy on the CIFAR-10 test set for ResNet-18 trained with combinations of SSAPT ($\mathcal{P}_0$, $\mathcal{P}_1$, $\mathcal{P}_2$) and AT ($\mathcal{T}_0$, $\mathcal{T}_5$).

Method / ϵ_{test}	$0/255$	$3/255$	$4/255$	$5/255$	$6/255$	$7/255$	$8/255$	$9/255$	$10/255$
$\mathcal{P}_0 \vert \mathcal{T}_0$	85.29	71.90	66.63	60.96	55.42	49.52	44.01	39.28	34.41
$\mathcal{P}_1 \vert \mathcal{T}_0$	84.67	72.67	67.91	62.61	56.91	51.46	45.91	40.27	35.88
$\mathcal{P}_2 \vert \mathcal{T}_0$	**85.37**	72.01	66.69	61.05	55.52	49.61	43.70	38.17	33.66
$\mathcal{P}_1 \vert \mathcal{T}_5$	83.78	72.75	68.20	63.32	58.25	52.85	47.37	42.24	37.93
$\mathcal{P}_2 \vert \mathcal{T}_5$	84.08	72.65	68.08	62.94	58.01	52.77	47.16	41.67	36.58
$\mathcal{P}_0 \vert \mathcal{T}_5$	84.61	**73.33**	**68.61**	**64.01**	**59.09**	**53.73**	**48.11**	**43.42**	**38.76**

C Additional results

C.1 Is there something special about rotation prediction?

Our experiments suggest the answer to be no. Refer to Table C3, where we compare swapping the task of predicting image rotation to the jigsaw task [2], where one scrambles the image parts into one of several pre-selected permutations and then predicts the nature of the permutation. We see from Table C3 that either the rotation or jigsaw task succeed in improving the robustness of the model in a similar way, though we can see that jigsaw is slightly inferior to rotation prediction.

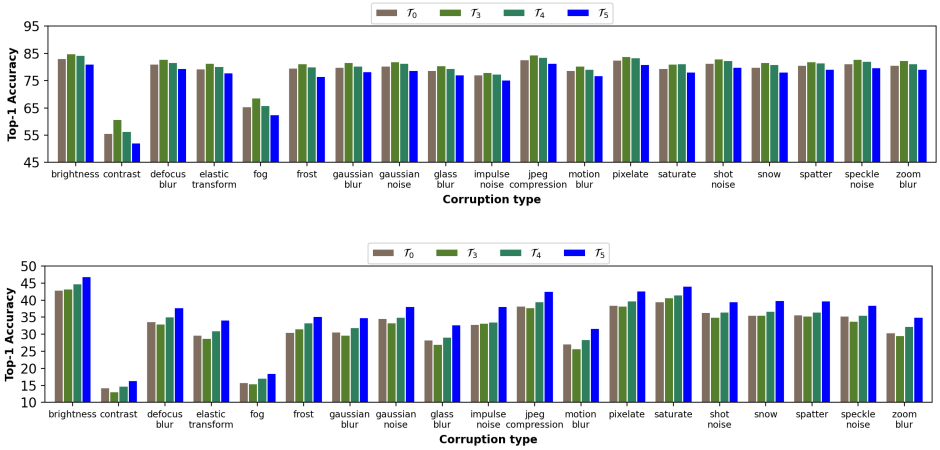

Figure C1: Comparison of various methods to incorporate self-supervision into adversarial training ($\mathcal{T}_0$, $\mathcal{T}_3$, $\mathcal{T}_4$, $\mathcal{T}_5$) for the various corruption types present in the corrupted CIFAR-10 dataset. (Top): TA, non-robust top-1 accuracy. (Bottom): RA, robust top-1 accuracy.

Table C3: Top-1 accuracy on the CIFAR-10 test set with ResNet-18 models trained with l_∞ norm and $\epsilon_{\text{train}} = 8/255$ evaluated at different values of ϵ_{test} used to generate the robust test set as we vary the SSL tasks that are used in tandem with AT.

Method / ϵ_{test}	$0/255$	$3/255$	$4/255$	$5/255$	$6/255$	$7/255$	$8/255$	$9/255$	$10/255$
Baseline	**85.46**	72.09	66.98	61.29	55.83	50.19	44.69	39.38	34.60
Rotation	84.61	**73.33**	**68.61**	**64.01**	**59.09**	**53.73**	**48.11**	**43.42**	**38.76**
Jigsaw	83.90	72.39	67.91	63.18	57.99	52.89	47.57	42.31	37.66

Table C4: Top-1 accuracy on the CIFAR-10 test set with ResNet-18 models trained with l_∞ norm and $\epsilon_{\text{train}} = 8/255$ evaluated at different values of ϵ_{test} used to generate the robust test set as we vary the training method and SSL involvement.

Method / ϵ_{test}	$0/255$	$3/255$	$4/255$	$5/255$	$6/255$	$7/255$	$8/255$	$9/255$	$10/255$
$\mathcal{T}_0$ [24]	85.16 (0.44)	72.73 (0.35)	67.76 (0.42)	62.55 (0.56)	57.06 (0.58)	51.47 (0.66)	46.00 (0.79)	40.82 (0.79)	35.83 (0.87)
$\mathcal{T}_0$ w/ rot aug.	72.54 (0.50)	61.90 (0.33)	57.82 (0.27)	53.70 (0.25)	49.38 (0.42)	45.07 (0.35)	40.68 (0.40)	36.38 (0.45)	32.19 (0.56)
$\mathcal{T}_1$	74.60 (0.51)	63.60 (0.44)	59.41 (0.27)	55.04 (0.29)	50.57 (0.27)	45.96 (0.36)	41.34 (0.45)	36.82 (0.49)	32.47 (0.46)
$\mathcal{T}_2$	74.51 (0.58)	63.10 (0.35)	58.67 (0.28)	54.27 (0.18)	49.67 (0.35)	45.16 (0.41)	40.63 (0.59)	36.05 (0.66)	31.79 (0.55)
$\mathcal{T}_3$	86.49 (0.50)	73.16 (0.38)	67.64 (0.47)	61.64 (0.59)	55.51 (0.66)	49.54 (0.72)	43.59 (0.77)	38.17 (0.96)	33.03 (0.98)
$\mathcal{T}_4$	85.90 (0.39)	73.95 (0.28)	68.82 (0.33)	63.27 (0.52)	57.65 (0.49)	51.85 (0.50)	46.23 (0.54)	40.77 (0.64)	35.76 (0.60)
$\mathcal{T}_5$	84.54 (0.53)	72.70 (0.40)	67.97 (0.51)	62.89 (0.50)	57.60 (0.55)	52.23 (0.57)	46.88 (0.54)	41.67 (0.47)	36.80 (0.40)

Table C5: Comparison of TA and RA of various models for $\mathcal{T}_0$ and $\mathcal{T}_5$ on the CIFAR-10 dataset.

MODEL	TA $\mathcal{T}_0$	TA $\mathcal{T}_5$	RA $\mathcal{T}_0$	RA $\mathcal{T}_5$
VGG-16	**76.84**	75.87	43.58	**44.35**
RESNET-18	**85.46**	84.61	44.69	**48.11**
RESNET-34	**86.04**	86.03	45.83	**47.94**
RESNET-50	**84.85**	82.97	46.00	**48.10**
WRN-40-2	**83.56**	83.01	46.43	**48.42**

Table C6: Comparison of ResNet-18 TA and RA for $\mathcal{T}_0$ and $\mathcal{T}_5$ on various datasets.

DATASET	TA $\mathcal{T}_0$	TA $\mathcal{T}_5$	RA $\mathcal{T}_0$	RA $\mathcal{T}_5$
CIFAR-10	**85.46**	84.61	44.69	**48.11**
CIFAR-100	**60.99**	60.09	24.20	**26.01**

C.2 Robustness to natural corruptions

In this section, we report the performance of the best SSL training methods on the CIFAR-10C dataset, which contains various common corruptions, e.g. Gaussian noise, blur [14]. Figure C1 shows the plots of the non-robust TA (Top) and robust RA (Bottom) Top-1 accuracies for l_∞-robust ResNet18 models trained with $\epsilon_{\text{train}} = 8/255$ (the plot for l_2-robust models can be found in the Supplement). From all of these examples we can see the following trend (which is similar to the l_2 trained models): adding the self-supervised loss ($\mathcal{T}_3$ and $\mathcal{T}_4$) helps the model improve its TA accuracy for the various common image corruptions — the clean accuracy goes from 78.36% mean accuracy over all corruptions to 80.23% and 79.15% for $\mathcal{T}_3$ and $\mathcal{T}_4$ trained models respectively. The clean accuracy drops from 78.36% mean accuracy to 76.52% when SSL task loss is added to PGD ($\mathcal{T}_5$). However, one can see the benefit of $\mathcal{T}_5$ when adversarial perturbations are added to natural corruptions as RA goes from 35.83% to 38.33% mean RA for l_2-robust models and from 32.20% to 36.22% mean RA for l_∞-robust models. In terms of RA, $\mathcal{T}_3$ and $\mathcal{T}_4$ are worse than $\mathcal{T}_0$ (in case of l_2-robust models) and $\mathcal{T}_3$ is worse than $\mathcal{T}_0$ for l_∞. Interestingly, even though the mean TA for both l_2 and l_∞ robust models is similar, l_2-robust models (trained using $\mathcal{T}_0$, $\mathcal{T}_4$, $\mathcal{T}_5$) outperformed l_∞ robust models.

C.3 Robustness improvement across datasets and models

Lastly, we compare the TA and RA of baseline AT ($\mathcal{T}_0$) and the self-supervised augmented ensemble adversarial training ($\mathcal{T}_5$) using: (a) various architectures on CIFAR-10 (Supp. materials), and (b) on various datasets using ResNet-18 (Supp. materials). In both of the tables, we observe the same trend we saw previously and see that ASSET ($\mathcal{T}_5$) improves model robustness across various deep learning architectures and datasets.