RoDINO: Boosting Empirical Robustness of Representations by Leveraging Modern Attacks

Ali Rasteh* **Sara Ghazanfari*** **Varuni Buereddy***
ar7655@nyu.edu sg7457@nyu.edu vb2386@nyu.edu

Electrical and Computer Engineering Department
New York University

Abstract

Recently, learning task-agnostic representations using foundation models has gotten great attention in computer vision. Rich representations learned using foundation models are showing impressive performance on several downstream tasks including classification, segmentation, Image retrieval, etc. On the other hand, as it's well known, Neural Networks are vulnerable to adversarial attacks. The vulnerability of foundation models to adversarial attacks harms the performance of the model to all of the downstream tasks, and therefore having a robust representation can have a huge impact on the robustness of all tasks. Considering this fundamental impact, in this project, we propose RoDINO (**Ro**bust **DINO**) which is a method to boost the empirical robustness of downstream tasks by leveraging PGD attack to generate adversary images and adversarially train DINO [1] which is a self-supervised representation learning model with Vision Transformers backbone. The code is available at `https://github.com/SaraGhazanfari/RoDINO`

1 Background

Self-Supervised Representation Learning Self-supervised representation learning is a recent direction of research that leverages millions of unlabeled images to learn representations without any further supervision. Most of the methods in this area exploit the Contrastive Learning approach which optimizes the parameters by forcing the representations of different views of the same image to be close to each other and far from other images in a batch. DINO [1] is a self-supervised representation learning method that uses a Vision Transformer as its backbone and unlike other methods doesn't exploit the contrastive learning approach, it uses a self-distillation approach as shown in Figure 1. x is the training image and by two different augmentation pipelines, x_1 and x_2 are generated and passed to the student and teacher models, and the cross entropy loss is applied on the representations of x_1 and x_2 and forces these two to be close to each other. Remarkably, both student and teacher models are the same model, the student model's parameters are updated after each iteration of the training however the teacher model's parameters are updated after each epoch is fully finished.

Projected Gradient Descent The PGD (Projected Gradient Descent) method is one of the traditional methods to generate attacks for neural networks which is based on gradient descent or more technically gradient ascent method. We can define the projected gradient descent as:

$$\delta^* = \arg\max_{\|\delta\|_p \leq \epsilon} (\mathcal{L}(h_\theta(x+\delta), y))) \tag{1}$$

The general algorithm of the attack is as follows:

$$\text{For N times Do: } \delta = \rho(\delta + \alpha \nabla_\delta \mathcal{L}(h_\theta(x+\delta), y)) \tag{2}$$

* The names are sorted alphabetically.

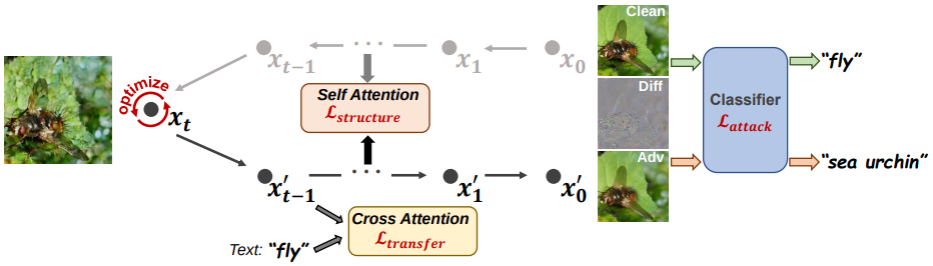


Figure 1: General framework of DiffAttack

Attack model	Clean Accuracy (%)	Adversarial Accuracy (%)
ViT (surrogate model)	84.1	9.4
Resnet	80.6	39.06
Vgg	76.68	40.1
Mobilenet	74.5	38
Inception	73.9	38.84

Table 1: Transferability of DiffAttack on other models: The adversarial images generated on the surrogate model are sent to different models like Resnet, VGG, Mobilenet and Inception.

In the above formula, the ρ is the projection into the ball for the corresponding norm used in the adversarial attack (for example in the ℓ_∞ it's a clipping). There is also another version of PGD which is called the steepest descent and works as follows:

$$\text{For N times Do: } \delta = \delta + argmax_{\|v\| \leq \alpha} v^T \nabla_\delta \mathcal{L}(h_\theta(x+\delta), y) \tag{3}$$

This version In this work, we are going to use the traditional PGD shown in equation 1 for attacking the representation space and also adversarially training the representations using the attacks. We will use ℓ_∞ as our measure for the attacks and defense process as the ℓ_∞ is the measure that is more consistent with the human eye perception.

Adversarial training In this project, we aim to adversarially train the DINO representation to make it more robust against adversarial attacks. The adversarial training using PGD attack could be formalized as follows:

$$\theta^* = \arg\min_\theta (\max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(h_\theta(x+\delta), y))) \tag{4}$$

As shown in equation 4 in adversarial training, in each batch we find the δ that maximizes the loss in that batch using the current set of parameters and change the input with that δ and then train the network with those set of inputs to minimize the loss and find the new θ. We continue this process until we reach some convergence point. In practice if you just feed the model with adversary inputs you can't get an acceptable natural accuracy, so usually we feed both the natural and adversary input to the network to train it in a balanced way. In DINO we can choose to train the model from scratch or fine-tune a pre-trained model.

Diffusion Attack In recent years, the field of machine learning has witnessed the evolution of diffusion models, a class of probabilistic generative models, which have proven to be remarkably effective in a wide array of applications. They have revolutionized tasks ranging from image generation to data denoising, attracting significant attention from researchers and practitioners alike. [2] explores the advances in diffusion models, with a particular focus on their unconventional and powerful application: crafting adversarial examples that are transferable and imperceptible. They perturb the image's latent in diffusion models, which is of good imperceptibility together with excellent transferability across various black-box models. Figure. 1 adopts a Stable Diffusion and leverages DDIM Inversion to convert the clean image into the latent space. The latent is optimized to deceive the classifier. The cross-attention maps are leveraged to "deceive" the diffusion model, and they use self-attention maps to preserve the structure. Given a clean image x and its corresponding label y, attackers aim to craft perturbations that can deviate the decision of a classifier F_θ (θ denotes

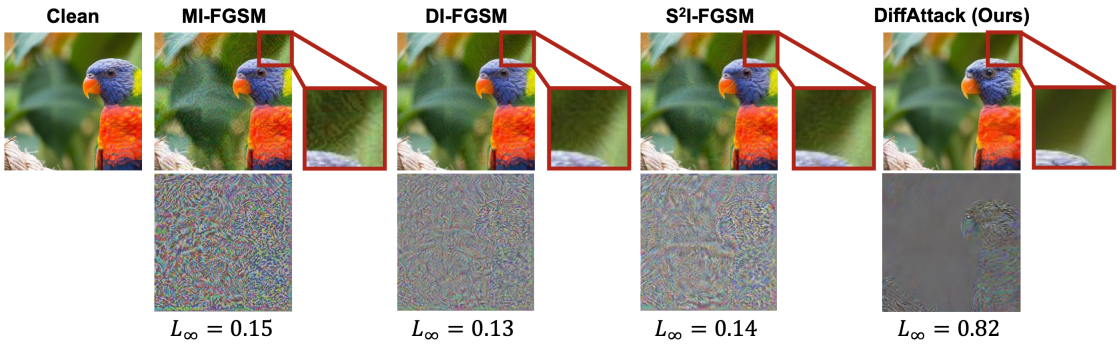


Figure 4: Adversarial perturbations crafted by DiffAttack compared to other methods. [2]

the model's parameters) from correct to wrong. x' denotes the crafted adversarial example. Diffattack algorithm satisfies the following properties:

- **Attack Capability**: DiffAttack ensures that the difference between the image reconstructed from the perturbed latent space and the original clean image is almost imperceptible. Interestingly, this difference image may contain valuable high-level semantic information, enhancing both the imperceptibility and transferability of the attack.
- **Transferability**: The algorithm leverages cross-attention maps, which establish a strong relationship between guided text and image pixels. This highlights the powerful recognition capabilities of pretrained diffusion models. However, distributing cross-attention uniformly across image pixels can disrupt the original semantic relationship, further contributing to the attack's success.
- **Structure Preservation**: Self-attention maps play a key role in preserving the structural features of the image. They ensure that the perturbed image retains its original structure, which is critical for the success of the adversarial attack.

Notably, the inversion strength parameter in DiffAttack serves as a trade-off between making the attack imperceptible and maximizing its potential to mislead classifiers. Through careful tuning, DiffAttack can achieve remarkable transferability across a diverse range of model architectures, including convolutional neural networks (CNNs), Transformers, and multi-layer perceptrons (MLPs).

Dataset We used the ImageNet-100 to generate adversarial examples using DiffAttack, and PGD. ImageNet-100 is a subset of ImageNet-1k Dataset from ImageNet Large Scale Visual Recognition Challenge 2012. It contains random 100 classes as specified in the Labels.json file. Train Contains 1300 images for each class. Validation contains 50 images for each class.

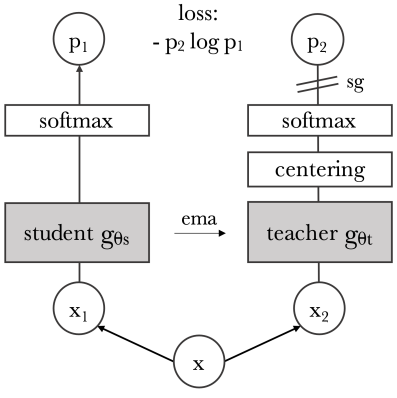


Figure 2: Dino Architecture.

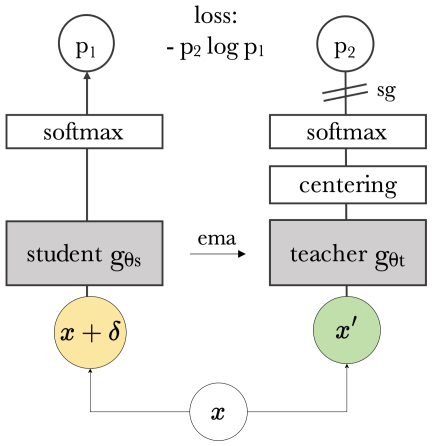


Figure 3: RoDino Architecture.

Dataset type	Job time (hours)	Adversarial Images Generated
Validation	28	1500
Train.X1	96	6200
Train.X2	96	5400
Train.X3	96	6500

Table 2: Time complexity of the DiffAttack for generating adversarial images on ImageNet-100K using 1x RTX 8000 GPU and 50GB RAM.

2 Method

In this section, we explain the details of our project. Initially, we mention why we considered DiffAttack as the base attack of adversarial training and the reason we substituted it with PGD. Second, we present the adversarial training of DINO considering its architecture which leads to RoDINO. Finally, we present the downstream tasks, Classification and KNN, employed to show the robustness of RoDINO and compare it with DINO.

2.1 DiffAttack in Adversarial Training

In this project, we aim to boost the robustness of DINO to ℓ_∞ attacks by leveraging adversarial training. In order to improve the efficiency of Adversarial training, we explored different ℓ_∞ attacks and finally decided to leverage the DiffAttack to generate the adversary data. First, most of the attacks generate adversary images for $\|\delta\|_\infty \leq 0.15$ despite the tiny perturbation budget, the perturbations are visible to human eyes, for DiffAttack [2] as shown in Figure 4 for much larger perturbation budget $\|\delta\|_\infty = 0.82$, the perturbations are fully imperceptible. Second, the efficiency of DiffAttack is not evaluated while leveraged as a mechanism to generate adversary data for training. Therefore we were interested in checking the robustness of the resulting model adversarially trained using DiffAttack. After choosing DiffAttack, we tried to evaluate its performance by doing experiments. One of the properties of DiffAttack that we analyzed is the transferability. To assess the transferability of DiffAttack, we generated the adversary images using the ViT-based model and then evaluated the robustness of other models to the generated attacks. The results are shown at Table 1. As presented in the table although the attack has the highest success rate on the base model (ViT), it has also an acceptable success rate on other models.

After working with DiffAttack in practice, we encountered during our experimentation that crafting adversarial examples using DiffAttack proved to be a time-intensive process, particularly when aiming to generate perturbations for the entire ImageNet100 dataset, as we wanted to the Adversarial Training, we needed to perform DiffAttack on Training data. Recognizing the computational demands, we made efforts to leverage High-Performance Computing (HPC) resources to expedite the process. However, despite these optimizations, the time complexity remained a significant challenge. The time complexity of the DiffAttack on the training and validation data is shown in table 2. In response to these computational constraints, we explored alternative approaches and opted to incorporate the Projected Gradient Descent (PGD) attack in our study. The decision to integrate PGD attack was driven by its reputation for efficiency in generating adversarial examples, especially in scenarios where time is a critical factor. This adjustment allowed us to navigate the challenges posed by the vast ImageNet-100K dataset, enabling a more pragmatic and feasible adversarial training process. The ensuing sections will detail our experiences with the PGD attack.

2.2 RoDINO: Adversarial Training on DINO

In order to boost the performance of DINO, we aim to perform the adversarial training. The DINO architecture and training scheme provides a proper infrastructure for performing Adversarial Training. As shown in Figure 1, the architecture has two branches, where the teacher and student models are the same models, but the parameters are updated differently. The parameters of student model are updated like the vanilla training after each iteration. However, the parameters of the teacher model are generated after each epoch and are updated using Exponential Moving Average (EMA) therefore holding the general knowledge of the whole training process till that point. To learn *good*

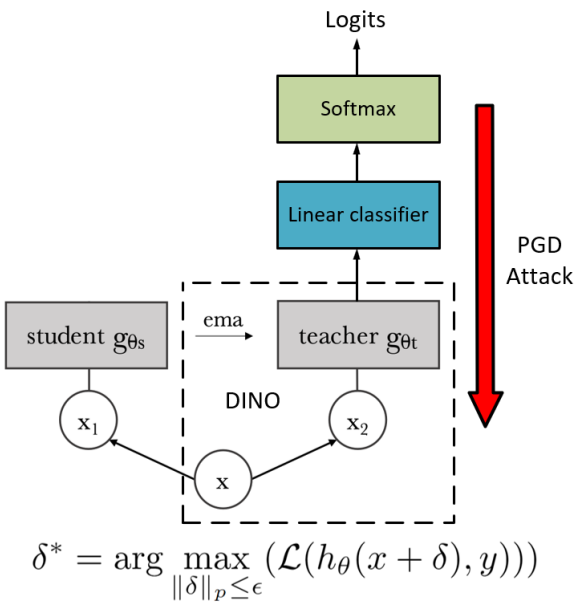


Figure 5: DINO + linear classifier architecture and attack methodology

representations of image x, two augmentation pipelines are structured, and two views x_1 and x_2 are generated from x and fed to the student and teacher branches and using the self-distillation paradigm the representation generated from the student branch is forced to have the same distribution as the representation of the teacher model: $min_{\theta_s} H(p_t(x), p_s(x))$ where s and t stand for student and teacher and H is the cross entropy loss: $H(a, b) = -a \log b$. Leveraging the structure of DINO we perform the adversarial training. The adversary image corresponding to x is generated using PGD attack substitutes the x_1 image which goes to the student branch and the teacher branch is left untouched. The modified architecture of DINO corresponding to RoDINO is shown in Figure 1.

2.3 Classification with RoDINO

DINO is a foundation model that gives us representations of the input space for different downstream tasks. In this project, we have used it for classification on the ImageNet-100K dataset. To do the classification we should use a classifier for the downstream tasks over the DINO backbone and as the representations given by DINO are very informative we are able to do it using just a linear classifier. Of course, one can use more complicated neural networks over DINO and get a better accuracy but the accuracy improvement probably won't be major. The architecture of the whole classifier including the DINO backbone and the linear classifier over DINO is shown in Figure 5. As you can see we have used the teacher sub-model of DINO to train the linear classifier (this is the default way that the authors of the DINO paper have used the DINO backbone over downstream tasks). We have used a linear layer and a softmax layer over that and finally, we get the logits. During the training phase of the linear classifier, the weights of the DINO backbone are frozen and not trained. For doing the PGD attack on the downstream tasks we compute the loss shown at the bottom of the figure and try to maximize the loss using δ. So the delta is computed on the whole network including both DINO and the linear classifier which is different from attacks that are generated on DINO adversary training (which is computed only on DINO).

2.4 KNN with RoDINO

K-Nearest Neighbors (KNN) is a simple and powerful algorithm used in machine learning for both classification and regression tasks, though it's more widely used for classification. At its core, KNN operates on a straightforward principle: it classifies a new data point based on how closely it resembles the existing data points in the training set. In the evaluation phase when a point is fed to the model, a representation of that point is generated and KNN looks around to find the 'K' nearest neighbors in the embedding space, where 'K' is a user-defined number. The algorithm then assigns the new point

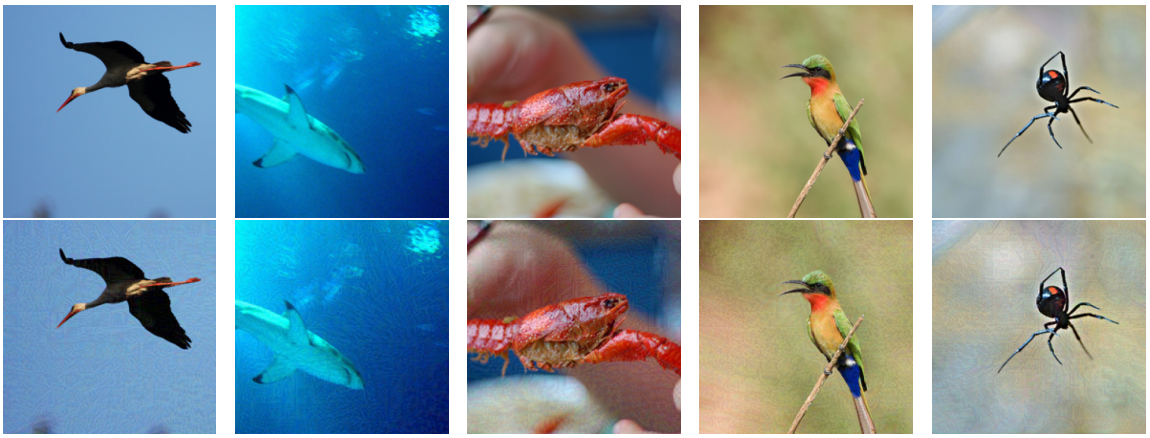


Figure 6: The first row of images represents the original images. The second row displays the corresponding adversarial images, generated using the Projected Gradient Descent (PGD-ℓ_∞ with ϵ = 0.03) attack on the Dino baseline model, which is misclassified when sent to a linear classifier.

Embedding	Natural Score	ℓ_∞-PGD			ℓ_2-PGD		
		0.01	**0.02**	**0.03**	**1.0**	**2.0**	**3.0**
DINO W Norm	**86.94**	28.00	10.02	3.62	NA	NA	NA
DINO	**86.94**	0.72	0.00	0.00	27.00	20.02	18.50
RoDINO-v1	79.36	43.20	20.48	7.64	59.64	57.04	56.28
RoDINO-v2	82.60	**50.12**	**32.26**	**17.08**	**61.34**	**59.42**	**58.76**

Table 3: Comparison between natural and adversary accuracy of DINO and RoDINO. RoDINO-v1: Adversary trained DINO on just the attack images for 1 epoch with 10 epoch trained linear classifier on the AT DINO. RoDINO-v2: Adversary trained DINO on both normal and attack images for 10 epochs with 10 epoch trained linear classifier on the AT DINO.

to the most common category among these neighbors. It's similar to finding the most popular vote or trend in the vicinity of the new point. This downstream task is appealing to our evaluation as our model RoDINO generates robust embeddings and without the need to add any more layers or any training, we can directly use the RoDINO representations to do the KNN task. In the experiments section we'll provide the results of our experiments.

3 Experiments

In this section, we present the experiments we performed on the RoDINO. The first set of experiments evaluates the accuracy and robustness of RoDINO on the classification and KNN downstream tasks. In a different setting, we assess the robustness of RoDINO representations against a direct attack on the representation.

3.1 Classification Task

In this part, we report the performance of the classification task using the linear classifier which was explained in the last sections. The results of the evaluation of DINO on the classification downstream task are shown in the table 3. The reference DINO model is downloaded and used from the repository provided by [1]. On the other hand, we have 2 versions of RoDINO. In RoDINO-v1 we adversary train DINO in just 1 epoch using just the attack images, which means that we only give attack images to the student model during fine-tuning of DINO. In RoDINO-v2 we adversary train DINO for 10 epochs using a mixture of normal and attack images fed to the student model. The linear classifier is separately trained over the reference DINO, RoDINO-v1, and RoDINO-v2. As you can see the

Embedding	Natural Accuracy 10-NN		Robust Accuracy 10-NN		Natural Accuracy 20-NN		Robust Accuracy 20-NN	
	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
DINO	**86.86**	**95.94**	50.38	66.6	**87.18**	**96.84**	51.08	69.08
RoDINO	83.46	90.97	**59.96**	**76.82**	83.88	93.04	**60.92**	**78.76**

Table 4: A comparison between DINO and RoDINO in terms of natural and robust accuracies. The natural accuracy of DINO is slightly better than RoDINO, however, there is a gap between the robust accuracy of the two methods and demonstrates the superiority of RoDINO embeddings in terms of robustness.

natural accuracy of DINO is better than both RoDINO versions however the natural accuracy of RoDINO-v2 is better than RoDINO-v1. On the other hand, we can see a significant improvement in the adversary accuracy of RoDINO-v1/v2 compared to DINO which means that the adversary training procedure has been very effective in making a more robust model. The performance of models is evaluated under different perturbation budgets of ℓ_∞ and ℓ_2 attacks. Also, the accuracy of RoDINO-v2 is better than v1 which means that fine-tuning the DINO with a mixture of normal and adversary images is a more effective way to make a robust DINO. In RoDINO-v2 we have defined a mixture of losses between the student model and teacher model which tries to force DINO to give a more similar representation for attack images compared to different augmented versions of the images.
In evaluating DINO and RoDINO we have always done normalization in the natural evaluations but always removed the normalization step while doing attacks because it's the only way that you can make sure the attacks are being performed correctly. In the DINO W Norm in the table, we have also reported the accuracy of the DINO reference model when being evaluated with normalization present on the attack images. As you can see the performance of DINO W normalization is better than DINO on ℓ_∞ which was expectable.

Figure 6 shows a comparison of original images and adversarial images generated by PGD-Linf that are classified correctly by RoDino but incorrectly by DINO.

3.2 K-Nearest Neighbors (KNN)

In this part, we present the evaluation we performed on DINO and RoDINO for the KNN task. The natural and robust accuracies are reported at Table 4. The natural accuracy of DINO is slightly better than RoDINO, however, there is a gap between the robust accuracy of the two methods and demonstrates the superiority of RoDINO embeddings in terms of robustness. Another observation is that results clearly demonstrate the accuracy-robustness tradeoff and the required cost we need to pay to provide better robustness.

3.3 Direct Attack to RoDINO

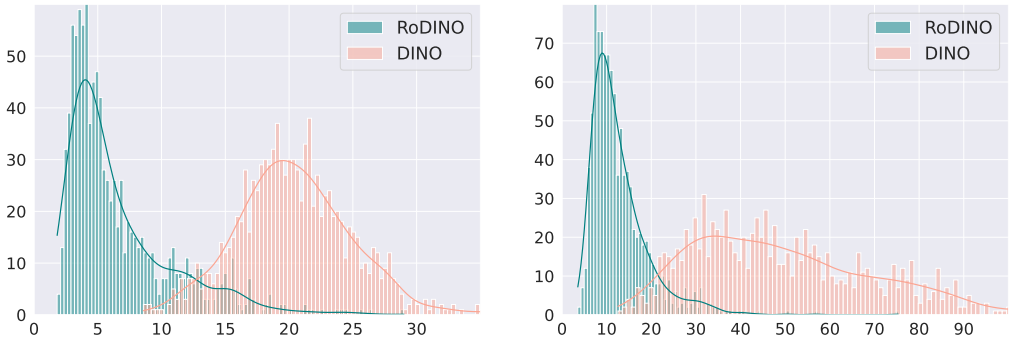
In this part, we perform a direct attack against the feature extractor model which is the source of the vulnerability for perceptual metrics by employing the ℓ_2-PGD ($\epsilon = 1.0$) and ℓ_∞-PGD [3] ($\epsilon = 0.01$) attack and the following MSE loss is used during the optimization:

$$\max_{\delta:\|\delta\|_2\leq\varepsilon} \mathcal{L}_{\text{MSE}}\left[f(x+\delta), f(x)\right] \tag{5}$$

The attack is performed on the validation set ImageNet-100 test set and against the DINO and RoDINO models. After optimizing the δ, the cosine distance metric is calculated between the original image and the perturbed image, which is defined as:

$$d(x, x+\delta) = 1 - S_c(f(x), f(x+\delta)) \tag{6}$$

Where $S_c(a,b) = \frac{a.b}{\|a\|_2\|b\|_2}$. The distribution of distances is shown in Figure 7. We can observe a shift in the mean of the distances for both histograms by comparing the RoDINO and DINO histograms.



(a) ℓ_2-PGD attack ($\epsilon = 1.0$)

(b) ℓ_∞-PGD attack ($\epsilon = 0.01$)

Figure 7: Direct Attack to DINO and RoDINO. The attack is optimized to maximize the MSE loss of the original image and perturbed image and the cosine distance is employed to calculate $d(x, x + \delta)$. The histogram of DINO shows a great shift from zero to large numbers, however, RoDINO managed to stay closer to zero.

The mean of the histograms corresponding to RoDINO are closer to zero compared with DINO histograms.

Conclusion & Future work

In this project, we propose **RoDINO**, a robust representation learning model based on DINO, which is a ViT-based Self-supervised learning model. The robustness of RoDINO is achieved by leveraging Adversarial Training and finetuning DINO using adversary data. The architecture facilitates the Adversarial Training by forcing the model to predict representation with similar distributions for both original and adversary data. The efficiency of our approach i.e., the robustness of RoDINO embeddings is evaluated by investigating two downstream tasks, classification and K-Nearest Neighbors. The comparison between DINO and RoDINO highlights the improved robustness and RoDINO's resilience against perturbations introduced by the PGD attack. To further investigate the robustness of RoDINO, a direct attack on the representations is performed and confirms on the enhanced robustness of RoDINO compared to DINO.

While our findings provide valuable insights into the effectiveness of adversarial training for enhancing DINO's robustness against PGD attacks, It would be interesting to perform evaluation across diverse attacks like AutoAttack and Blackbox attacks. Moreover, the robustness of RoDINO can be explored for other downstream tasks including segmentation, depth estimation, etc. To boost the performance of adversarial training, more sophisticated adversarial attacks can be employed to provide more robustness for the representations.

References

[1] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9650–9660, 2021.

[2] Jianqi Chen, Hao Chen, Keyan Chen, Yilan Zhang, Zhengxia Zou, and Zhenwei Shi. Diffusion models for imperceptible and transferable adversarial attack. *arXiv preprint arXiv:2305.08192*, 2023.

[3] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.