Learning Transferable Adversarial Perturbations

Anonymous Author(s)
Affiliation
Address
email

Abstract

While effective, deep neural networks (DNNs) are vulnerable to adversarial attacks. In particular, recent work has shown that such attacks could be generated by another deep network, leading to significant speedups over optimization-based perturbations. However, the ability of such generative methods to generalize to different test-time situations has not been systematically studied. In this paper, we therefore investigate the transferability of generated perturbations when the conditions at inference time differ from the training ones in terms of target architecture, target data, and target task. Specifically, we identify the mid-level features extracted by the intermediate layers of DNNs as common ground across different architectures, datasets, and tasks. This lets us introduce a loss function based on such mid-level features to learn an effective, transferable perturbation generator. Our experiments demonstrate that our approach outperforms the state-of-the-art universal and transferable attack strategies.

1 Introduction

In recent years, deep neural networks (DNNs) have achieved great success in a wide range of applications [1, 2, 3, 4]. However, DNNs have been demonstrated to be vulnerable to adversarial examples [5] crafted by adding imperceptible perturbations to clean images. In particular, two broad categories of attacks have been studied. The first one consists of iterative algorithms [5, 6, 7], which optimize the perturbation for each instance, and thus tend to be computationally expensive. The second one encompasses generative methods [8, 9, 10], which train a deep network to produce perturbations. As such, attacking the target network only involves a forward pass through the generator, typically resulting in much faster attacks than iterative methods. However, speed is not the only important factor to assess the strength of an attacker; its ability to generalize to different situations is also key to its success.

In this paper, we therefore study the transferability of perturbations obtained with generative methods. Specifically, we investigate the transfer of such perturbations when the conditions at inference time differ from the training ones in terms of **(i)** target *architecture*, e.g., the generator was trained to attack a VGG-16 but the target network is a ResNet152; **(ii)** target *data*, e.g., the generator was trained using the Paintings dataset but the test data comes from ImageNet; **(iii)** target *task*, e.g., the generator was trained to attack an image recognition model but faces an object detector at test time. To the best of our knowledge, for generative methods, our work constitutes the first attempt at transferability across tasks, and only [11] has studied generalization across architectures and data, by introducing a loss function acting on the relative class probabilities of the attacked and unattacked examples.

Here, by contrast, we improve the transferability of a perturbation generator across architectures, data, and tasks by exploiting the mid-level features of DNNs. The key motivation behind our approach is our observation that the mid-level features extracted by DNNs with different architectures, different data, or for different tasks bear strong similarities. This is illustrated in Figure 1, where we visualize the features extracted by different backbones and for different tasks, and thus different datasets, using the method of [12]. Our analysis suggests that adversarial perturbations that significantly affect the

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

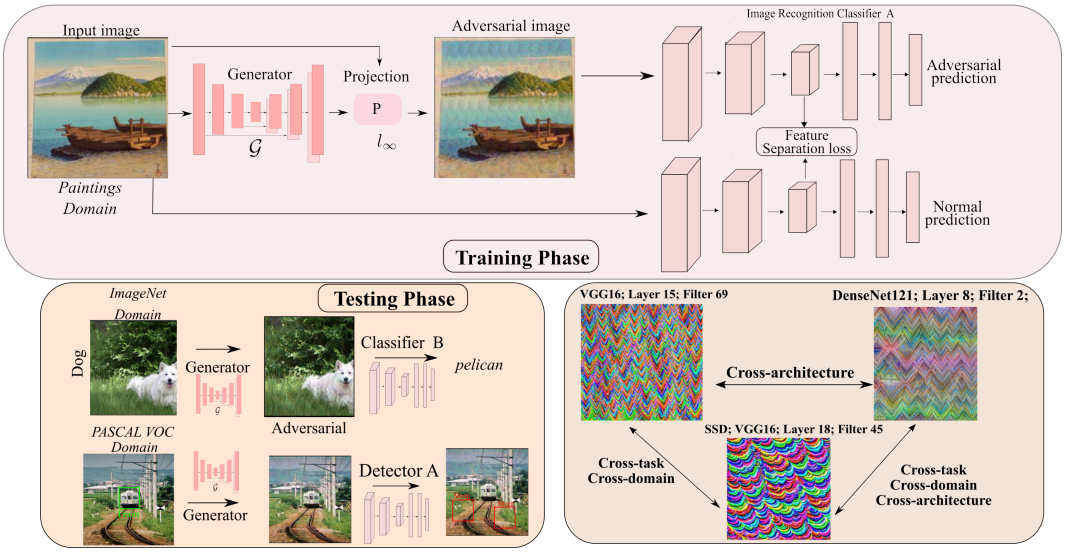


Figure 1: **Learning transferable perturbations.** We observe that mid-level features are common across architectures and tasks, and thus propose to exploit them to train a perturbation generator by maximizing the relative distance between normal and perturbed features. We show that such a generator is effective even in the presence of a different model, dataset, or task at test time.

mid-level features of one sample in one architecture also affect them in a different architecture, even for a different task and with different data.

We therefore propose to train a perturbation generator by maximizing the distance between the normal features of a sample and their adversarial counterparts extracted in the intermediate layers of a pretrained classifier. The resulting perturbations are then transferable across architectures, datasets and tasks because they similarly affect the mid-level features of the corresponding filters in the target setup. The perturbed mid-level features are then propagated to the top layers of the network, and, as a result, lead to incorrect predictions.

Contributions. Our contributions can be summarized as follows: 1. We identify the intermediate features of CNNs as common ground across different architectures, different data distributions and different tasks. 2. We introduce an approach that exploits such features to learn an effective, transferable perturbation generator. 3. We systematically investigate the effect of target architecture and target data distribution on the transferability of adversarial attacks. Our experiments demonstrate that our approach yields higher fooling rates than the state-of-the-art universal [8] and transferable [11] attacks. We will make our code publicly available upon acceptance of the paper.

2 Related Work

Below, we review the recent works on adversarial attacks, with a focus on generator-based approaches.

Adversarial Attacks. Adversarial attacks were first investigated in [5] to identify the vulnerability of modern deep networks to imperceptible perturbations in the context of image classification. Since then, several attack strategies have been studied, including single-step fast gradient descent [6, 13], and computationally more expensive optimization-based attacks, such as CW [14], JSMA [15], and others [16, 17, 7].

While the above methods are image-dependent, the existence of Universal Adversarial Perturbation (UAP) was first shown in [18], considering the task of learning a single perturbation that can fool a classifier independently of the input image. Such a UAP was iteratively updated based on its effectiveness to move the individual data samples across the decision boundary. Other UAPs based on the computation of the singular vectors of the Jacobian matrices of feature maps have been studied in [19]. In parallel to UAPs, several works have shown that iterative adversarial attack strategies could be transferred across architectures [20, 21, 22]. In particular, the recent work of [23] proposed

to improve targeted attack transferability across architectures by training auxiliary classifiers at each intermediate layer to attack a CNN's feature maps. The above-mentioned methods, however, either are computationally expensive at inference time [16, 7, 15, 14, 21, 23], or suffer from low transferability rates [18, 20, 22]. Furthermore, most of the attacks aiming for transferability [18, 19, 23] strongly depend on the availability of data from the target domain. By contrast, we introduce an efficient generative method to produce adversarial perturbations that generalize not only across architectures, but also across training data and across tasks.

Generator-based Attacks. Generative adversarial perturbations (GAP) were first introduced in [10]. In particular, [10] showed that a generator network can be used to craft a UAP that transform an input image to an image-dependent perturbation. Similarly, [24] introduced the advGAN generative framework to learn to produce adversarial perturbations, and further proposed to make use of distillation to perform black-box attacks. Based on the observation that the effectiveness of these two methods strongly depends on the availability of data from the target domain during training, [11] introduced the relativistic cross-entropy loss, which was shown to better generalize across datasets. Furthermore, [8] proposed to generate UAPs by transforming the generator's output using a regional norm layer that enforces perturbation homogeneity. While effective, the above-mentioned works explicitly rely on the classification boundary of the attacked model, which we will show to tend to make them overfit to the source data. By contrast, we identify the mid-level features as a more robust signal shared across not only different architectures and datasets, but even across different tasks. Our experiments will showcase the superiority of exploiting this information over the losses used in previous approaches to train a transferrable perturbation generator.

3 Methodolgy

Let $\mathbf{x}_i \in \mathbf{R}^{H \times W \times 3}$ be a color image of size $H \times W$, and $\mathbf{y}_i$ be the associated ground-truth label. Furthermore, let f denote the task-related convolutional neural network that takes $\mathbf{x}$ as input and extracts, at layer $l = \{l_j\}_{j=1}^{L}$, a feature map $f_\ell(\mathbf{x}_i) \in \mathbf{R}^{N_\ell \times D_\ell}$, which we assume to be reshaped in matrix form, such that $N_\ell = H_\ell \times W_\ell$, with H_ℓ and W_ℓ the spatial dimensions of the feature map, and D_ℓ its number of channels.

Our goal is to train a generator $\mathcal{G}$ that produces a perturbation $\delta_i \in \mathbf{R}^{H \times W \times 3}$, which, when added to the clean image $\mathbf{x}_i$, results in predicting a different label from $\mathbf{y}_i$. To this end, we feed the input image $\mathbf{x}_i$ to the generator to synthesize an unbounded adversarial image $\mathcal{G}(\mathbf{x}_i)$, which is then clipped to be within an ϵ bound of $\mathbf{x}_i$ under the ℓ_∞ norm. Let $\hat{\mathbf{x}}_i$ be the adversarial image obtained after such a clipping of $\mathcal{G}(\mathbf{x}_i)$. In contrast to [10, 11], which rely on the final classification boundary of the task-related network f to train the generator with a cross-entropy-based loss, we exploit the mid-level features of f. Specifically, we maximize the L_2 distance between the normal feature map $f_\ell(\mathbf{x}_i)$ and adversarial feature map $f_\ell(\hat{\mathbf{x}}_i)$ at layer l using a feature separation loss term defined as

$$\mathcal{L}_{feat}(\mathbf{x}_i, \hat{\mathbf{x}}_i) = ||f_\ell(\mathbf{x}_i) - f_\ell(\hat{\mathbf{x}}_i)||_F^2 \ , \tag{1}$$

where $\| \cdot \|_F$ denotes the Frobenius norm.

The overall training scheme of our perturbation generator is provided in Algorithm 1. In practice, we observe the feature separation loss of Eq. 1 to encourage the generator to learn low-level, quasi-imperceptible structured patterns that affect that activations of a few filters while either suppressing the other ones or leaving them unaffected. More importantly, these perturbations are highly transferable.

Reason for transferability. Given the simplicity of our formulation, a natural question that arises is what affects the transferability of perturbations among CNNs. To understand this, we analyze the internal workings of a CNN, whose filters in each layer encode different levels of information. In particular, in Figure 2, we visualize the filters of various CNNs, sampled at different layers, using the technique of [12], which maximizes the mean activation of the each filter. This allows us to discover common characteristics across the different architectures. On one hand, the bottom filters, close to the input image, extract color and edge information, as shown in the left block of Figure 2. On the other, the top-level filters, close to the output layer, shown in the right block, are more focused on the object representation, and thus more task specific. By contrast, the mid-level filters learn more nuanced features, such as textures, and therefore tend to display similar patterns across architectures, datasets tasks. As a consequence, and as indicated by our experiments, attacking the initial layers requires large perturbation strengths. Furthermore, while attacking the top-level features works well

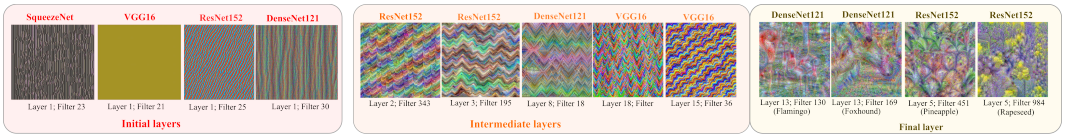
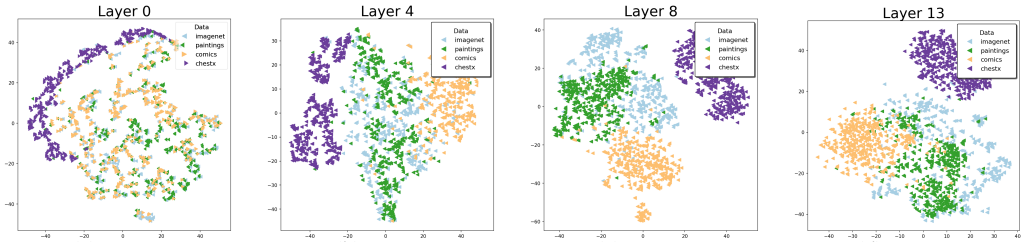


Figure 2: **Filter visualization.** The filters in the mid-level layers of different CNNs follow similar activation patterns, also common across tasks, whereas those near the classification layer focus on object-level features and are thus more task-specific.



(a) Layer 0 (b) Layer 4 (c) Layer 8 (d) Layer 13

Figure 3: **Feature visualization.** We visualize the features extracted at 4 different layers of DenseNet121 for different datasets using t-SNE.

in the white box scenario, it does not transfer well to different data or architectures, because the features are already too architecture- and task-specific. Attacking mid-level features thus comes as a natural choice, which we will show to yield highly transferable perturbations.

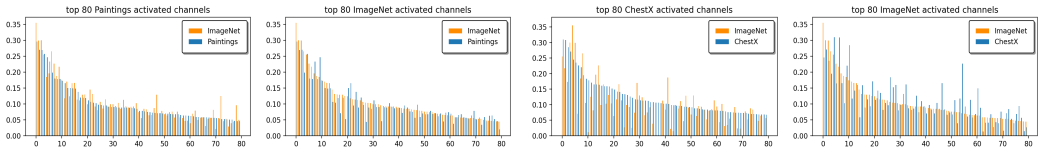
Let us now focus on understanding the reason for the transferability when the target data is unavailable to the attacker. In Figure 3, we plot t-SNE visualizations of different datasets at 4 layers of DenseNet121. The proximity of the features extracted from the Comics or Paintings datasets, containing images from a few classes particularly focused on humans, to those extracted from ImageNet suggests these datasets as good candidates to train a generator that will fool an ImageNet-trained network (as shown in Tables 1 and 2). By contrast, the ChestX dataset presents a larger domain gap. Nevertheless, we will show that, albeit for a drop in fooling rate, our use of mid-level features still make a ChestX-trained generator reasonably effective on ImageNet-based classifiers. Furthermore, in Figure 4, we plot the top 80 activations of ImageNet, Paintings and ChestX at layer 8 of DenseNet121. This shows that the peaks of the ImageNet dataset (orange) are more strongly correlated to those of the Paintings dataset (left) than to those of ChestX (right). This is confirmed by our experiments, where a Paintings-trained generator is more effective than a ChestX-trained one. In short, while mid-level features lead to generator transferability, the resulting effectiveness remains affected by the similarity of the data.

Algorithm 1 Training a transferable adversarial perturbation generator

Input: $\mathbf{x}_i$: clean images; f: pretrained classifer; ϵ: perturbation ℓ_∞ bound

Initialize generator $\mathcal{G}$; Load classifier f and freeze its parameters;

repeat
  Get a clean image $\mathbf{x}_i$ and feed $\mathbf{x}_i$ to f to obtain $f_\ell(\mathbf{x}_i)$;
  Generate an unbounded adversarial image $\mathcal{G}(\mathbf{x}_i)$ and clip it within an ϵ bound of $\mathbf{x}_i$ to get $\hat{\mathbf{x}}_i$;
  Pass the adversarial image to f to obtain $f_\ell(\hat{\mathbf{x}}_i)$;
  Compute the feature separation loss $\mathcal{L}_{feat}$;
  Compute the gradient of $\mathcal{L}_{feat}$ w.r.t. the weights of $\mathcal{G}$ and update these weights using Adam.
until convergence
return trained Generator $\mathcal{G}^*$



(a) **Paintings vs. ImageNet** (b) **Chestx vs. ImageNet**

Figure 4: **Top activated channels.** Top-80 activated channels for ImageNet (orange) in comparison to Paintings in **(a)**, and to ChestX in **(b)**, for layer 8 of DenseNet121 using 500 images from each dataset. The blue bars are more correlated to the orange ones in the Paintings case, thereby achieving higher transferability rates than using ChestX. Best viewed in color and zoomed in.

Gen. Training (data)	**Classifier**	**VGG16** [31]	**ResNet152 [32]**	**Inception-v3 [34]**	**DenseNet121 [33]**	**SqueezeNet11 [35]**	**Average**
		GAP [10] / CDA [11] / Ours					
ImageNet (1.2M)	VGG-16	99.1 / **99.9** / 99.4	49.9 / 53.6 / **69.4**	40.1 / 40.7 / **48.4**	55.1 / 58.7 / **86.8**	67.0 / 67.2 / **88.9**	53.0 / 55.0 / **73.4**
	ResNet152	86.0 / 97.2 / **99.1**	98.1 / 99.6 / **99.7**	55.4 / 65.9 / **72.3**	85.5 / 95.4 / **98.7**	72.9 / 81.9 / **88.0**	74.9 / 85.0 / **89.5**
	DenseNet121	94.8 / 94.1 / **99.5**	89.1 / 78.3 / **98.3**	55.4 / 56.3 / **82.4**	99.6 / 99.6 / 99.5	81.1 / 79.0 / **94.2**	80.1 / 76.9 / **93.6**
	Inception-v3	88.2 / 97.0 / **98.7**	83.1 / 81.9 / **89.7**	96.9 / 98.5 / **99.6**	89.9 / 93.2 / **95.9**	90.9 / **92.0** / 91.3	88.0 / 91.0 / **93.9**
	SqueezeNet	90.1 / 90.0 / **96.1**	54.1 / 53.9 / **77.6**	49.0 / 50.0 / **72.5**	67.4 / 69.1 / **89.4**	99.8 / 99.8 / 99.7	65.1 / 65.8 / **83.9**
	Average						72.3 / 74.7 / **85.5**

Table 1: **White-box and standard black-box settings.** We set $\epsilon = 10$ and report the fooling rate (in %) on 5K samples from the ImageNet val-set. The best result in each section is shown in bold.

4 Experiments

We evaluate the effectiveness of our attack strategy in diverse settings. Below, we first discuss our experimental setup and then compare our attacks with the state-of-the-art generator-based ones: CDA [11], RHP [8], and GAP [10]. To further demonstrate the generalizability of our approach, we also report its performance on adversarially-trained models and on the SSD [25] object detector with 4 different backbones. We discuss the experimental details in the supplementary material.

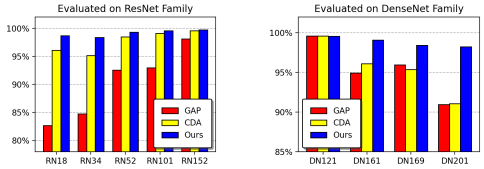
Datasets. To train the generator, similarly to [11], we use data from either ImageNet [26], Comics [27], Paintings [28] or ChestX [29] as source domain, containing 1.2M, 40K, 80K, and 8K images, respectively. We then randomly select 5000 images from the ImageNet [26] validation set as target domain to evaluate the transferability of our attacks. As in [11, 18, 10], we report the fooling rate and the absolute difference in top-1 error between before and after the attack. The fooling rate is the percentage of images for which the label is changed after the attack.

Models. For the target models, we make use of the publicly available PyTorch [30] versions of VGG-16 [31], VGG-19 [31], ResNet152 [32], DenseNet121 [33], Inception-V3 [34] and Squeezenet [35] pretrained on ImageNet. We chose this family of networks to understand the deeper impact of transferability across diverse architectures. As in [11], to evaluate transfer across datasets, we use ChestXNet [29] pretrained on ChestX. For the perturbation generators, we use the same architectures as in [11, 10]. To train them, we use the Adam optimizer with a learning rate of $2e^{-4}$ and a batch size of 16. We train our generator for a maximum of 80K iterations.

Attack settings. We perform attacks in four settings: **1.** White-box attacks, where the attacker has access to the exact target model and target data distribution; **2.** Standard black-box attacks, where the attacker has access to a substitute model from a different family of architectures trained on the target data and to the target data itself; **3.** Strict back-box attacks, where the attacker also uses a substitute model trained on the target data, but *without* having access to the target data itself; **4.** Extreme black-box scenario, we perform attacks without any knowledge of the target model or the target data.

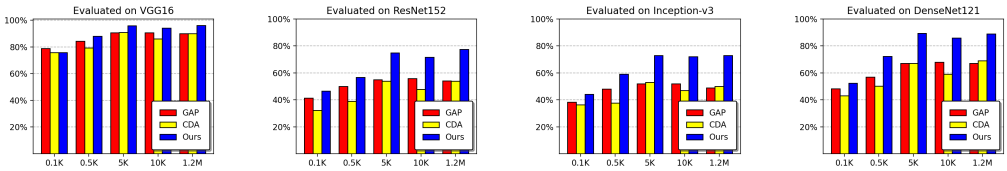
4.1 Transferability to Unknown Target Model

Let us first study the standard white box and black-box settings. In this experiment, we train the generator with target data containing 1.2M ImageNet training samples, using one of the 5 ImageNet pretrained models as target architecture. We then evaluate the transferability of the attack to the remaining 4 models.



(a) Trained on ResNet152 (b) Trained on DenseNet121

Figure 5: **Transferability within the same family.** We report the fooling rates when transferring attacks to other networks within the same family. The generators were trained with ResNet152, resp. DenseNet121, with $\epsilon = 10$.



(a) VGG-16 (b) ResNet152 (c) Inceptionv3 (d) DenseNet121

Figure 6: **Limited data setup.** We report the fooling rates obtained using varying amounts of ImageNet training data. The generators were trained with SqueezeNet [35] and $\epsilon = 10$, and transferred to other networks. The x-axis represents the amount of training data.

Table 1 compares the effectiveness of our attacks to the CDA and GAP ones. We outperform these baselines on average by 13.2 and 11.8 percentage points (pp) (i.e., absolute difference of fooling rates), respectively. The differences are particularly pronounced when the generator is trained with SqueezeNet. Furthermore, in the DenseNet121 case, the difference between CDA and Ours is 16.7pp.

A similar trend can be seen when transferring attacks within the same family of networks. For instance, Figure 5 (a) shows the transferability when the generator is trained with ResNet152 and evaluated on other networks of the same family, namely, ResNet18, ResNet34, ResNet52, and ResNet101. Figure 5 (b) shows the results for similar experiments on the DenseNet family. Our method performs on average 1.47pp, 8.9pp better than CDA and GAP in the case of ResNets, which demonstrates that our feature separation loss generalizes well for attacks within the same family. Importantly, the performance gap between Ours and the baselines increases with the difference in depth from training – from 0.08pp on DenseNet121 to 7.3pp on DenseNet201 w.r.t. GAP.

In Figure 6, we further study the transferability of attacks as the number of samples from the target data to train the generator varies. Our method (blue bar) consistently maintains high fooling rates, even for a small number of training samples, whereas GAP (red bar) and CDA (yellow bar) do not. Note that all methods tend to saturate when around 5K samples are available, with our method consistently outperforming the baselines.

4.2 Transferability to Unknown Target Data

Let us now turn to the strict black-box setting, where the target data is unavailable to the attacker, who then relies on a different dataset to train the generator. Note that, here, the attacker still has access to a substitute model trained on the target data, therefore we also study the impact of transferability to an unknown target model. Following [11], we consider two synthetic datasets, Comics and Paintings, depicting objects similar to the ImageNet ones, and one dataset, ChestX, containing a completely different type of data, thus suffering from a large domain gap.

Table 2 compares the results of our approach with those of the baselines. Our method yields clearly superior results for all 3 datasets. For instance, with Comics, Ours outperforms GAP and CDA on average by 12.0pp and 14.1pp, respectively, and with Paintings, by 20pp and 19pp, respectively. Note that, while we still outperform the baselines when using the ChestX data, by around 17pp on average, the fooling rates of all methods drop significantly in this challenging scenario. Nevertheless, altogether, these results evidence that one can learn to generate adversarial examples with high

Gen. Training (data)	Discriminator (pretrained on ImageNet)	VGG16	ResNet152	Inception-v3	DenseNet121	SqueezeNet	Average
		GAP [10] / CDA [11] / Ours					
Comics (40K)	VGG-16	**99.7** / 99.4 / 99.4	56.1 / 59.7 / **75.5**	47.2 / 46.8 / **60.2**	67.7 / 70.8 / **92.1**	69.6 / 81.7 / **92.8**	68.1 / 71.7 / **84.0**
	ResNet152	74.4 / 90.6 / **99.1**	99.5 / 98.9 / **99.7**	48.3 / 56.3 / **71.6**	60.0 / 89.5 / **98.5**	74.9 / 78.2 / **87.0**	68.1 / 82.7 / **91.2**
	DenseNet121	96.2 / 92.2 / **97.9**	**97.4** / 86.7 / 95.6	80.9 / 66.1 / **81.4**	99.0 / 98.6 / **99.4**	78.7 / 75.9 / **91.9**	90.5 / 83.9 / **93.3**
	Inception V3	84.0 / 73.7 / **99.0**	97.8 / 52.3 / **90.7**	99.7 / 76.4 / **99.7**	89.1 / 60.2 / **96.8**	80.6 / 83.5 / **93.3**	90.3 / 69.2 / **95.9**
	SqueezeNet	87.5 / 90.5 / **96.0**	53.4 / 56.6 / **78.1**	51.0 / 51.4 / **74.8**	67.1 / 76.7 / **89.6**	99.6 / 99.7 / 99.7	71.7 / 75.0 / **87.7**
	Average						78.3 / 76.5 / **90.4**
Paintings (80K)	VGG-16	99.5 / **99.7** / 99.1	37.5 / 42.5 / **66.8**	38.2 / 39.7 / **46.6**	50.3 / 53.5 / **83.4**	60.7 / 64.3 / **85.1**	46.7 / 50.0 / **70.5**
	ResNet152	81.6 / 77.8 / **98.2**	97.4 / 97.9 / **99.6**	58.4 / 48.7 / **58.8**	79.8 / 68.4 / **97.1**	72.0 / 75.9 / **81.6**	72.3 / 85.7 / **93.4**
	DenseNet121	85.5 / 86.3 / **98.3**	79.9 / 80.5 / **97.2**	50.5 / 51.2 / **76.9**	98.6 / 98.6 / **98.9**	73.3 / 74.7 / **92.4**	72.3 / 73.1 / **91.2**
	Inception V3	77.0 / 81.1 / **98.6**	55.9 / **96.7** / 88.7	80.1 / **99.7** / 99.4	69.5 / 88.4 / **95.6**	75.4 / 76.9 / **90.8**	69.4 / 85.7 / **93.4**
	SqueezeNet	81.2 / 82.3 / **95.2**	47.2 / 46.8 / **73.8**	45.4 / 45.4 / **68.5**	56.8 / 58.3 / **86.6**	99.1 / 99.5 / **99.7**	58.1 / 57.6 / **81.0**
	Average						63.8 / 66.9 / **84.0**
ChestX (10K)	VGG-16	73.4 / 75.2 / **96.0**	22.3 / 22.8 / **60.8**	25.2 / 25.8 / **41.6**	26.5 / 26.8 / **68.5**	46.4 / 44.4 / **65.8**	32.2 / 32.5 / **55.4**
	ResNet152	40.7 / 35.4 / **50.5**	27.0 / 20.7 / **64.1**	26.5 / 21.0 / **27.3**	26.1 / 22.5 / **33.0**	45.1 / 42.7 / **60.4**	27.7 / 23.7 / **39.2**
	DenseNet121	42.4 / 64.5 / **66.3**	24.6 / 39.8 / **46.4**	27.5 / 33.4 / **32.7**	40.1 / 51.0 / **75.4**	51.9 / 59.1 / **59.4**	31.0 / 41.2 / **46.6**
	Inception V3	37.6 / 46.6 / **71.6**	23.6 / 27.4 / **55.3**	31.5 / 36.9 / **71.0**	26.5 / 32.5 / **60.0**	47.5 / 57.1 / **66.7**	27.8 / 33.4 / **54.0**
	SqueezeNet	51.7 / 48.6 / **79.9**	29.3 / 29.4 / **50.2**	30.4 / 31.9 / **43.3**	36.0 / 34.2 / **60.6**	76.0 / 77.2 / **96.1**	37.2 / 36.9 / **55.0**
	Average						31.2 / 33.5 / **50.1**

Table 2: **Transferability in the strict black-box setting with no access to target data.** We set $\epsilon = 10$ and report the fooling rate for 5K samples from the ImageNet validation set.

fooling rates despite not having access to the target data, even by using data from a completely different domain. Furthermore, the resulting generator generalizes particularly well to the unseen target domain when trained with our feature separation loss.

Discriminator (Model)	Gen. Training (data)	VGG16 [31]	ResNet152 [32]	Inception-v3 [34]	DenseNet121 [33]	SqueezeNet [35]	Average
		GAP [10] / CDA [11] / Ours					
ChestXNet (trained on ChestX)	Comics40K	65.9 / 58.1 / **75.4**	49.3 / 42.5 / **50.2**	54.3 / 47.8 / **55.7**	61.1 / 49.6 / **76.0**	69.6 / 68.9 / **75.8**	60.0 / 53.4 / **66.7**
	Comics10K	61.2 / 44.8 / **77.6**	44.4 / 30.3 / **56.7**	52.3 / 36.2 / **61.2**	54.9 / 39.1 / **77.3**	64.1 / 55.9 / **79.3**	55.4 / 41.2 / **70.4**
	Paintings10K	74.3 / 50.9 / **77.0**	51.1 / 35.9 / **59.5**	57.4 / 44.5 / **60.7**	70.8 / 42.0 / **78.4**	76.3 / 61.8 / **80.9**	66.0 / 47.0 / **71.3**
	Paintings80K	74.4 / 64.0 / **77.1**	51.8 / 51.4 / **57.5**	56.3 / 48.5 / **59.4**	72.8 / 62.7 / **77.7**	77.9 / 69.4 / **80.1**	66.6 / 59.2 / **70.4**
	ChestX10k	46.0 / 51.6 / **67.1**	30.8 / 30.3 / **47.7**	36.5 / 34.2 / **49.6**	37.4 / 36.8 / **64.6**	53.9 / 55.4 / **71.0**	40.9 / 41.7 / **60.0**
	Average	62.3 / 56.0 / **74.8**	45.4 / 38.1 / **54.3**	49.8 / 43.8 / **57.3**	57.4 / 48.1 / **74.8**	66.7 / 63.9 / **77.4**	56.3 / 50.0 / **67.7**

Table 3: **Extreme cross-domain transferability analysis using a generator trained with ChestXNet and different datasets.** We set $\epsilon = 10$ and report the fooling rates on 5K ImageNet val-set samples.

4.3 Extreme Cross-Domain Transferability

We further study the transferability of adversarial attacks in the extreme scenario where neither the target architecture nor the target data are available, and where even the classifier used to learn the generator was trained on data that differs from the target domains. To this end, we use ChestXNet [29] pretrained on ChestX dataset as classifier. ChestXNet is similar to the DenseNet121 architecture but was trained to classify a lung X-Ray image into one of multiple classes using the sigmoid cross-entropy loss.

We train the generator with different datasets and evaluate its transferability in Table 3. Despite the challenging nature of this setting, our method still yields satisfying fooling rate; 67.7pp on average vs 85.5% in the standard black-box scenario. Furthermore, our approach outperforms the state-of-the-art CDA by 17.9% on average.

4.4 Transferability to Robust Models

In this section, we study the transferability of our generators to adversarially trained models. To this end, we use the state-of-the-art three ensembles of adversarially trained models [36], namely, Inc-v3$_{ens3}$, Inc-v3$_{ens4}$ and IncRes-v2$_{ens4}$, on ImageNet. In addition to GAP and CDA, we compare our method to UAP and RHP, which constitute the state-of-the-art universal attacks in terms of effectiveness against defenses. We train the generator with the naturally trained, i.e., undefended, Inception-v3. As shown in Table 4, Ours achieves the overall best performance across all competing methods. This suggests that, although adversarially trained models learn robust features, there remains a vulnerable subspace common across architectures.

4.5 Cross-Task Transferability Analysis

To demonstrate the transferability of our approach across tasks, we attack object detectors using perturbation generators trained with image classifiers. Specifically, we choose the SSD framework

Method	Training	Inc-v3$_{ens3}$	Inc-v3$_{ens4}$	IncRes-v2$_{ens4}$	Average
Inc-v3	UAP	1.00	1.80	1.88	1.59
	RHP	32.5	31.6	24.6	29.5
Inc-v4	RHP	27.5	26.7	21.2	25.1
IncRes-v2	UAP	1.88	1.74	1.96	1.86
	RHP	29.7	29.8	26.8	28.7
GAP + ImageNet		39.8	35.30	32.6	35.9
CDA + ImageNet		43.38	40.68	37.64	40.57
Ours + ImageNet		**49.22**	**46.04**	**39.02**	**44.76**

Table 4: **Transferability to adversarially trained models**. We report the percentage of increase in top-1 error for generators trained with the natural Inception-v3 model (with $\epsilon = 16$) and evaluated on 3 ensembles of adversarially trained models.

Gen. Training (data)	Discriminator (Trained on ImageNet)	VGG16 [31]	ResNet50 [32]	EfficientNet	MobileNet-v3	Average
		GAP [10] / CDA [11] / Ours				
-	No Attack	68.12	66.08	61.07	55.44	62.68
Comics (40K)	VGG-16	19.7 / 26.7 / **9.40**	15.3 / 23.9 / **12.9**	**12.9** / 18.6 / 14.2	**8.50** / 13.1 / 10.1	13.9 / 20.6 / **11.6**
	ResNet152	33.2 / 28.6 / **13.3**	23.4 / 18.9 / **10.9**	29.1 / 17.5 / **12.6**	20.9 / 16.7 / **8.30**	26.6 / 20.4 / **11.3**
	DenseNet121	21.2 / 27.1 / **15.2**	18.8 / 21.6 / **18.1**	18.9 / 21.5 / **16.8**	12.8 / 16.7 / **11.3**	17.9 / 21.7 / **15.3**
	Inception-v3	34.9 / 27.6 / **23.3**	28.5 / 25.3 /**24.9**	28.7 / **18.6** / 21.5	19.6 / **16.9** / 18.7	28.0 / 22.1 / 22.1
	SqueezeNet	29.7 / 33.7 / **18.1**	25.1 / 29.8 / **16.5**	19.5 / 25.8 / **19.1**	15.3 / 22.2 / **12.1**	22.4 / 27.9 / **16.4**
	Average					21.8 / 22.5 / **15.8**
Paintings (80K)	VGG-16	24.5 / 22.7 / **10.3**	23.7 / 21.2 / **13.4**	19.5 / 16.6 / **13.8**	14.6 / **11.2** / 13.5	20.6 / 17.9 / **12.7**
	ResNet152	26.4 / 36.8 / **22.6**	23.1 / 24.5 / **19.8**	19.8 / 22.6 / **18.5**	17.3 / 20.7 / **12.5**	21.7 / 26.2 / **18.3**
	DenseNet121	33.2 / 33.8 / **12.7**	25.1 / 25.0 / **13.0**	24.8 / 26.0 / **13.0**	21.1 / 21.0 / **7.50**	26.0 / 26.4 / **11.6**
	Inception-v3	26.8 / 33.3 / **16.5**	25.7/ 28.6 / **18.4**	18.9 / 27.7 / **13.7**	19.6 / 23.9 / **13.6**	18.8 / 28.4 / **15.6**
	SqueezeNet	36.6 / 35.0 / **16.8**	29.5 / 28.8 / **14.7**	27.6 / 25.8 / **14.9**	20.4 / 20.3 /**9.20**	28.5 / 27.5 / **13.9**
	Average					23.3 / 25.3 / **14.4**
ImageNet (1.2M)	VGG-16	19.2 / 18.0 / **7.50**	20.2 / 19.3 / **8.60**	10.1 / 10.1 / **12.4**	9.20 / 9.50 / **5.30**	14.7 / 14.2 / **8.50**
	ResNet152	27.4 / 17.6 / **8.50**	17.8 / 13.4 / **8.60**	14.0 / 10.8 / **6.50**	14.1 / 7.20 / **3.90**	18.3 / 12.3 / **6.90**
	DenseNet121	21.6 / 19.2 / **6.50**	15.6 / 17.2 / **7.80**	14.0 / 15.8 / **7.50**	12.6 / 12.4 / **4.10**	16.0 / 16.2 / **6.40**
	Inception-v3	**12.9** / 13.9 / 15.4	15.6 / **14.7** / 18.1	13.0 / **10.7** / 14.5	12.7 / **10.7** / 12.2	13.6 / **12.5** / 15.1
	SqueezeNet1	26.2 / 27.1 / **13.7**	22.0 / 22.8 / **11.2**	17.6 / 19.4 / **12.0**	14.1 / 15.4 / **7.20**	16.5 / 15.2 / **9.60**
	Average					16.5 / 15.3 / **9.6**

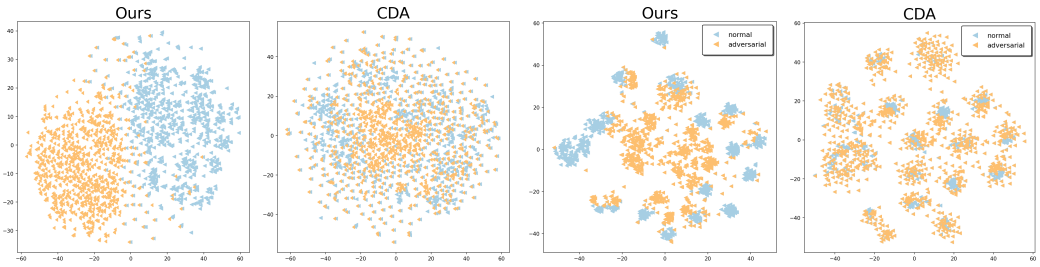
Table 5: **Transferability across tasks with access to neither the target data nor the target model.** We set $\epsilon = 16$ and report the fooling rates for 4952 samples from the PASCAL VOC test-set.

with 4 different backbones, namely, VGG16, ResNet50, EfficientNet, and MobileNet-v3 pretrained on PASCAL VOC [37]. In Table 5, we report the mAP on the PASCAL VOC test-set, containing 4952 images, for perturbation generators trained with different image classifiers and with $\epsilon = 16$. The performance of the SSD detector on clean images is 68.1, 66.1, 61.1 and 55.4 using VGG16, ResNet50, EfficientNet and MobileNetv3 as backbone, respectively. Our attacks significantly decrease these score and yield lower mAPs than the baselines in most cases. These results highlight, for example, that a generator trained on the synthetic Comics dataset with an image classifier can fool an SSD detector trained on a different domain, with a different architecture, and for a different task. As evidenced by the comparisons with CDA and GAP, learning to generate perturbations that affect the mid-level features is more effective than focusing on the classification boundary.

4.6 More Analysis

To further analyze our method, we visualize the resulting perturbed images using t-SNE plots. Specifically, in Figure 7(a), we provide t-SNE plots obtained using the final latent representation of 1000 normal (blue) and attacked (yellow) ImageNet images for the strict black-box and extreme black-box scenarios. In both cases, our method yields features that clearly separate the normal and adversarial images with large margins, whereas CDA fails to do so. This evidences that disrupting the mid-level features propagates to the final layers of a different network, leading to incorrect prediction.

In Figure 8, we visualize the resulting adversarial images with $\epsilon = 10$. The perturbations look like repeating patterns corresponding to some mid-level filter patterns. Note that RHP [8] explicitly enforces perturbation homogeneity using a predefined pattern, such as vertical or horizontal patterns. Our filter visualizations for multiple networks reveals that the bottom layers learn such horizontal and vertical stripes. While we tried training the generator to perturb such layers, we observed that larger ϵ strength are required in this case. Overall, our results evidence that attacking mid-level filters that have strong correlations across architectures consistently improves the transfer rates.



(a) **Train**: Paintings on SqueezeNet; **Eval:** DenseNet121 (b) **Train**: Comics10K on ChestXNet; **Eval:** ResNet152

Figure 7: **t-SNE visualizations.** We show t-SNE plots for 1000 normal (blue) images and their adversarial (yellow) counterparts in **(a)** the strict black-box setting and **(b)** the extreme black-box setting. More visualizations can be found in the supplementary material.

VGG16 **DenseNet121** **Inception-v3** **Squeezenet**

Figure 8: **Qualitative Results.** Visualization of resulting adversarial images for generators trained with ImageNet dataset. We observe that the pertubations are structured and repeating patterns that affects the mid-level filters. Best viewed in color and zoom.

Limitations. In our experiments, we choose a single layer l at a time. To this end, we sweep over each block of layers, typically around 5 on average for the studied architectures. We empirically found that the best-performing layer is independent of the choice of the target model to attack. For example, a generator trained with a feature separation loss on the conv4-2 layer in VGG16 transfers to all attack settings at inference time, avoiding repeated sweeps and computational costs. For all our experiments, we set the layer l to attack to conv4-2, layer3, mixed6b, denseblock8, and fire10 for VGG16, ResNet152, Inception-v3, DenseNet121, and SqueezeNet, respectively. We report additional results with generators trained at different layers in the supplementary material. Nevertheless, we have no guarantees that such layers truly are optimal; in particular, better results might be achievable by attacking ensembles of layers, but this would require tuning the layer combination.

5 Conclusion

In this paper, we have explored the use of mid-level features in conjunction with generative networks to learn transferable perturbations. We have shown that a generator trained with feature separation loss can successfully fool models across architectures and tasks, and that a proxy dataset from a different domain can be leveraged to learn effective perturbations. Our experiments demonstrate that our method outperforms the state-of-the-art attacks across a wide range of architectures and tasks. We have limited our experiments to undefended models due to the unavailability of publicly available robust models across architectures but will study this in future work. Furthermore, we believe that a deeper analysis of learned filter banks with respect to changes in architectures can shed light on building better black-box models.

6 Broader Impact

Understanding the strength of adversarial attacks is critical to building robust defenses in the future. The practical implications of adversarial examples in safety-critical applications, such as autonomous driving and biometrics, are huge. Our extensive set of experiments highlights the vulnerability of modern networks to adversarial attacks even in the extreme black-box case, which has a high potential for societal impact. We hope that our work will motivate the community to further study the internal workings of neural networks in the context of adversarial attacks, and in particular exploit such analysis to design effective defense mechanisms.

References

[1] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[3] Shaoqing Ren, Kaiming He, Ross B Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *NIPS*, 2015.

[4] Shuiwang Ji, Wei Xu, Ming Yang, and Kai Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2012.

[5] 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.

[6] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[7] 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.

[8] Yingwei Li, Song Bai, Cihang Xie, Zhenyu Liao, Xiaohui Shen, and Alan L Yuille. Regional homogeneity: Towards learning transferable universal adversarial perturbations against defenses. *arXiv preprint arXiv:1904.00979*, 2019.

[9] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

[10] Omid Poursaeed, Isay Katsman, Bicheng Gao, and Serge Belongie. Generative adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4422–4431, 2018.

[11] Muzammal Naseer, Salman Khan, Muhammad Haris Khan, Fahad Khan, and Fatih Porikli. Cross-domain transferability of adversarial perturbations. In *33rd Conference on Neural Information Processing Systems (NeurIPS)*, volume 32. NEURAL INFORMATION PROCESSING SYSTEMS (NIPS), 2019.

[12] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

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

[14] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[15] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 372–387. IEEE, 2016.

[16] Yinpeng Dong, Fangzhou Liao, Tianyu Pang, Hang Su, Jun Zhu, Xiaolin Hu, and Jianguo Li. Boosting adversarial attacks with momentum. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9185–9193, 2018.

[17] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[18] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[19] Valentin Khrulkov and Ivan Oseledets. Art of singular vectors and universal adversarial perturbations. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8562–8570, 2018.

[20] Konda Reddy Mopuri, Utsav Garg, and R Venkatesh Babu. Fast feature fool: A data independent approach to universal adversarial perturbations. *British Media Vision Conference*, 2017.

[21] Nathan Inkawhich, Wei Wen, Hai Helen Li, and Yiran Chen. Feature space perturbations yield more transferable adversarial examples. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7066–7074, 2019.

[22] Konda Reddy Mopuri, Phani Krishna Uppala, and R Venkatesh Babu. Ask, acquire, and attack: Data-free uap generation using class impressions. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 19–34, 2018.

[23] Nathan Inkawhich, Kevin J Liang, Lawrence Carin, and Yiran Chen. Transferable perturbations of deep feature distributions. *arXiv preprint arXiv:2004.12519*, 2020.

[24] Jamie Hayes and George Danezis. Learning universal adversarial perturbations with generative models. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 43–49. IEEE, 2018.

[25] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[27] Cenk Bircanoglu. Painter by number. *https://www.kaggle.com/cenkbircanoglu/comic-books-classification*, 2017.

[28] Painter by number. *https://www.kaggle.com/c/painter-by-numbers/data*, 2017.

[29] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

[31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[33] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[34] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[35] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016.

[36] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[37] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

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 Section 4.6

(c) Did you discuss any potential negative societal impacts of your work? [Yes] We discussed in Section 6

(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? [N/A]

(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)? [No] The code is not submitted with the current submission. We will release the code upon acceptance.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] We briefly mentioned experimental setup in Section 4. In addition, we elaborate the setup for each experiment in the supplementary

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]

(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]

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] We have cited all benchmark datasets that we used in the paper.

(b) Did you mention the license of the assets? [N/A]

(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]