Appendix

A Details of the Settings in the Experiment

A.1 Models

In order to avoid overfitting of the AITL model and ensure the fairness of the experimental comparison, we use completely different models to conduct experiments during the training and evaluation of the AITL model.

During the process of training AITL, we utilize totally eleven models to provide the attack success rate corresponding to the input transformation, including ten normally trained models (*i.e.*, ResNet-50 [23], Xception [8], DenseNet-201 [26], VGG-19 [49], MobileNetv2-1.0 [47], MobileNetv2-1.4 [47], ResNeXt-101 [72], SENet-101 [25], EfficientNetB4 [54] and EfficientNetv2S [55]) and one adversarially trained model (*i.e.*, AdvInceptionv3 [58]). All models are publicly available[6,7]. In the process of training Adaptive Image Transformation Learner, we utilize the MobileNetv2-1.0 as the source white-box model (as the grey model in Fig. 2) to generate the adversarial examples, and regard the other models as the target black-box models (as the other colorful models in Fig. 2) to evaluate the corresponding attack success rates, *i.e.*, q_{asr}.

During the process of evaluation, we use seven normally trained models (*i.e.*, Inceptionv3 (Incv3) [52], Inceptionv4 (Incv4) [51], Inception-ResNetv2 (IncResv2) [51], ResNetv2-101 (Resv2-101) [24], ResNetv2-152 (Resv2-152) [24], PNASNet [37] and NASNet [78]), three adversarially trained models (*i.e.*, Ens3Inceptionv3 (Ens3-Incv3), Ens4Inceptionv3 (Ens4-Incv3) and EnsInceptionResNetv2 (Ens-IncResv2) [58]). All models are publicly available[8, 7]. In addition, another eight stronger defense models are also used to evaluate the generated adversarial examples, including HGD [35], R&P [70], NIPS-r3[9], Bit-Red [73], JPEG [21], FD [39], ComDefend [29] and RS [28].

A.2 Baselines

We utilize several input-transformation-based black-box attack methods (*e.g.*, DIM [71], TIM [16], SIM [36], CIM [74], Admix [62]) to compare with our method. By default, we incorporate these methods into MIFGSM [15], *i.e.*, using the formula of Eq. (5). Besides, we also combine these input-transformation-based methods together to form the strongest baseline, called Admix-DI-SI-CI-MIFGSM (ADSCM). We also compare our AITL with recently proposed AutoMA [75].

In addition, we use a random selection method instead of the AITL model to choose the combination of image transformations used in the attack, which

[6] `https://keras.io/api/applications`
[7] `https://github.com/wowowoxuan/adv_imagenet_models`
[8] `https://github.com/tensorflow/models/tree/master/research/slim`
[9] `https://github.com/anlthms/nips-2017/tree/master/mmd`

is denoted as `Random` method. From the experiments in Sec. 4.3, we know that the geometry-based image transformations are more effective than most color-based image transformations (except `Scale`) to improve the transferability of adversarial examples. So we exclude these color-based image transformations (except `Scale`) from the transformation candidates, and finally choose 12 transformations in `Random` method, including `Admix`, `Scale`, `Admix-and-Scale`, `Crop`, `Resize`, `Rotate`, `ShearX`, `ShearY`, `TranslateX`, `TranslateY`, `Reshape`, `Cutout`.

For the hyperparameters used in baselines, the decay factor μ in MIFGSM [15] is set to 1.0. The transformation probability p in DIM [71] is set to 0.7. The kernel size k in TIM [16] is set to 7. For a fair comparison of different methods, we control the number of repetitions per iteration in all methods to 5, *i.e.*, the number of scale copies m in SIM [36] is set to 5, m_1 and m_2 in Admix [62] are set to 1 and 5, respectively, m in AutoMA [75] is set to 5, and N in our AITL is also set to 5.

A.3 Image Transformation Operations

Inspired by [11,12] and considering the characteristic in the task of adversarial attack, we totally select 20 image transformation operations as candidates, including `Admix`, `Scale`, `Admix-and-Scale`, `Brightness`, `Color`, `Contrast`, `Sharpness`, `Invert`, `Hue`, `Saturation`, `Gamma`, `Crop`, `Resize`, `Rotate`, `ShearX`, `ShearY`, `TranslateX`, `TranslateY`, `Reshape`, `Cutout`. In this section, we introduce each transformation operation in detail, and give the range of magnitude towards each operation.

Geometry-based Operations. Many image transformation operations are based on affine transformation. Assuming that the position of a certain pixel in the image is (x, y), the operation of affine transformation can be formulated as:

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{pmatrix}}_{A} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix}, \tag{26}$$

where A is the affine matrix, and (x', y') is the position of the pixel after transformation.

- **Rotation.** The affine matrix in the `Rotation` operation is:

$$\begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{27}$$

  where θ is the angle of rotation. In implementation, we set $\theta \in [-30°, 30°]$.

- **ShearX.** The operation of `ShearX` is used to shear the image along x-axis, whose affine matrix is:
$$\begin{pmatrix} 1 & a & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{28}$$
where a is used to control the magnitude of shearing. In implementation, we set $a \in [-0.5, 0.5]$.
- **ShearY.** The operation of `ShearY` is used to shear the image along y-axis, whose affine matrix is:
$$\begin{pmatrix} 1 & 0 & 0 \\ a & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{29}$$
where a is used to control the magnitude of shearing. In implementation, we set $a \in [-0.5, 0.5]$.
- **TranslateX.** The operation of `TranslateX` is used to translate the image along x-axis, whose affine matrix is:
$$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \tag{30}$$
where a is used to control the magnitude of translating. In implementation, we set $a \in [-0.4, 0.4]$.
- **TranslateY.** The operation of `TranslateY` is used to translate the image along y-axis, whose affine matrix is:
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & a \\ 0 & 0 & 1 \end{pmatrix}, \tag{31}$$
where a is used to control the magnitude of translating. In implementation, we set $a \in [-0.4, 0.4]$.
- **Reshape.** We use the operation of `Reshape` to represent any affine transformation, which has a complete 6 degrees of freedom. In implementation, we set $a_{11}, a_{22} \in [0.5, 1.5]$ and $a_{12}, a_{13}, a_{21}, a_{23} \in [-0.5, 0.5]$.
- **Resizing.** For the operation of `Resizing`, the original image with the size of $299 \times 299 \times 3$ is first randomly resized to $h \times w \times 3$, where $h, w \in [299, 330]$, and then zero padded to the size of $330 \times 330 \times 3$. Finally, the image is resized back to the size of $299 \times 299 \times 3$.
- **Crop.** The operation of `Crop` randomly crops a region of $h \times w \times 3$ size from the original image, where $h, w \in [279, 299]$, and then resizes the cropped region to the size of $299 \times 299 \times 3$.

Color-based Operations.

- **Brightness.** The operation of `Brightness` is used to randomly adjust the brightness of the image with a parameter $\alpha \in [0.5, 1.5]$ to control the magnitude. $\alpha = 0$ refers to a black image and $\alpha = 1$ refers to the original image.

- **Color.** The operation of `Color` is used to randomly adjust the color balance of the image with a parameter $\alpha \in [0.5, 1.5]$ to control the magnitude. $\alpha = 0$ refers to a black-and-white image and $\alpha = 1$ refers to the original image.
- **Contrast.** The operation of `Contrast` is used to randomly adjust the contrast of the image with a parameter $\alpha \in [0.5, 1.5]$ to control the magnitude. $\alpha = 0$ refers to a gray image and $\alpha = 1$ refers to the original image.
- **Sharpness.** The operation of `Sharpness` is used to randomly adjust the sharpness of the image with a parameter $\alpha \in [0.5, 1.5]$ to control the magnitude. $\alpha = 0$ refers to a blurred image and $\alpha = 1$ refers to the original image.
- **Hue.** The operation of `Hue` is used to randomly adjust the hue of the image with a parameter $\alpha \in [-0.2, 0.2]$ to control the magnitude. The image is first converted from RGB color space to HSV color space. After adjusting the image in the H channel, the processed image is then converted back to RGB color space.
- **Saturation.** The operation of `Saturation` is used to randomly adjust the saturation of the image with a parameter $\alpha \in [0.5, 1.5]$ to control the magnitude. The image is first converted from RGB color space to HSV color space. After adjusting the image in the S channel, the processed image is then converted back to RGB color space.
- **Gamma.** The operation of `Gamma` performs the gamma transformation on the image with a parameter $\alpha \in [0.6, 1.4]$ to control the magnitude.
- **Invert.** The operation of `Invert` inverts the pixels of the image, *e.g.*changing the value of pixel from 0 to 255 and changing the value of pixel from 255 to 0.
- **Scale.** The operation of `Scale` is borrowed from SIM [36], which scales the original image x with a parameter $m \in [0, 4]$:

$$\tilde{x} = \frac{x}{2^m}. \tag{32}$$

Other Operations.

- **Admix.** The operation of `Admix` is borrowed from Admix [62], which interpolates the original image x with another randomly selected image x' as follows:

$$\tilde{x} = x + \eta \cdot x', \tag{33}$$

  where $\eta = 0.2$ is used to control the magnitude of transformation.
- **Admix-and-Scale.** Since the operation of `Admix` changes the range of pixel values in the image, the processed image is likely to exceed the range of $[0, 255]$. So we combine the operation of `Admix` and `Scale` as a new operation of `Admix-and-Scale`.
- **Cutout.** The operation of `Cutout` cuts a piece of 60×60 region from the original image and pads with zero.

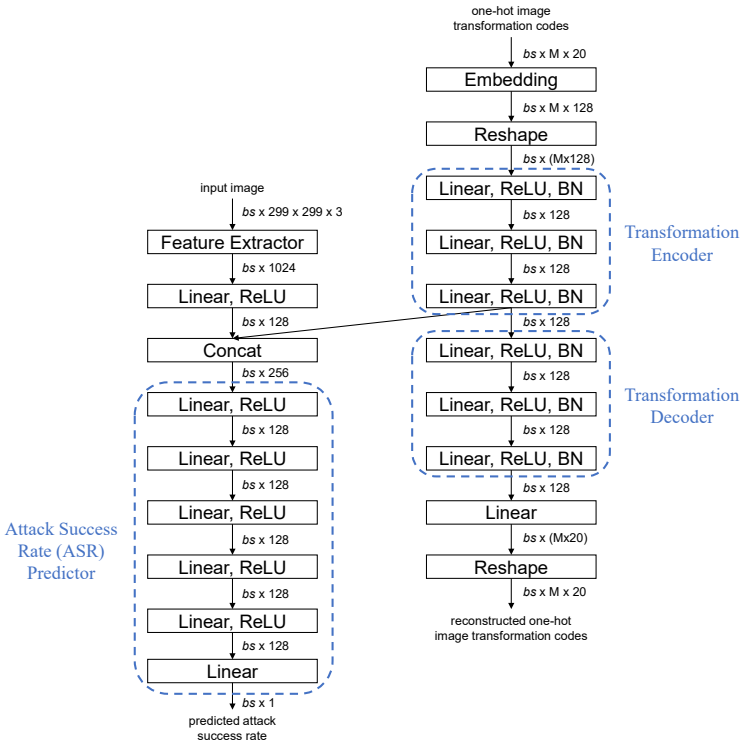


Fig. 5: The specific structure of the Adaptive Image Transformation Learner. The feature extractor utilizes the pre-trained MobileNetv2-1.0 [47] as initialization and is further finetuned together with the whole structure. The image features are extracted from the `Global_Pool` layer. The *bs* represents the batch size and M represents the number of image transformation operations in each combination

A.4 The detailed structure of AITL

The specific structure of the AITL network is shown in Fig. 5. The feature extractor utilizes the pre-trained MobileNetv2-1.0 [47] as initialization and is further finetuned together with the whole structure. The image features are extracted from the `Global_Pool` layer. The *bs* represents the batch size and M represents the number of image transformation operations in each combination.

B Discussion

The works most relevant to ours are AutoMA [75] and ATTA [68]. In this section, we give a brief discussion on the differences between our work and theirs.

B.1 Comparison with AutoMA.

Both AutoMA [75] and our AITL utilize a trained model to select the suitable image transformation for each image to improve the transferability of generated adversarial examples. The differences between the two works lie in the following aspects: 1) AutoMA adopts a reinforcement learning framework to search for a strong augmentation policy. Since the reward function is non-differentiable, Proximal Policy Optimization algorithm [48] as a trade-off solution, is utilized to update the model. Differently, we design an end-to-end differentiable model, which makes the optimization process easier to converge, leading to better image transformation combinations for various images. 2) AutoMA only takes the single image transformation as augmentation, while we further consider serialized combinations of different image transformations, so that more diverse image transformation choices can be taken during the generation of adversarial examples. 3) More kinds of image transformations are considered in the candidate set in our method (20 in our AITL *vs.* 10 in AutoMA). Extensive experiments in Sec. 4.2 show that our method significantly outperforms AutoMA, which demonstrates the superiority of our AITL.

B.2 Comparison with ATTA.

Both ATTA [68] and our AITL use transformed images to generate the adversarial examples, but there are also significant differences between the two. ATTA directly models the image transformation process through a pixel-to-pixel network to generate the transformed image. Since the solution space is quite large ($H*W*C$), it may be difficult to approach the optimal solution. On the contrary, our AITL turns to seek the optimal combinations of various existing transformations. The solution space of our method is reduced sharply to the number of image transformation operations ($\sim$20), so the optimization difficulty is well controlled and thus more tractable. The detailed experimental comparison and analysis between ATTA and our AITL are provided in Appendix C.7.

C Additional Experiments

C.1 Ablation Study

In this section, we analyze of effects of the number N of repetitions of the image transformation combination used in each attack step and the number M of image transformation operations in each combination.

The effect of the number N. In order to alleviate the impact of the instability caused by the randomness in image transformation on the generated adversarial examples, many existing methods [36,62,74] repeat the image transformation many times in parallel during each step of the attack. We change the number N of repetitions of the image transformation combination used in each attack step,

Table 4: The attack success rates under different number of repetitions of the image transformation combination during each attack step, *i.e.*, N in Fig. 1. The adversarial examples are crafted on Incv3. * indicates the white-box model

(a) The evaluation against 7 normally trained models

	Incv3*	Incv4	IncResv2	Resv2-101	Resv2-152	PNASNet	NASNet
AITL (N=5)	99.8	95.8	94.1	88.8	90.1	94.1	94.0
AITL (N=10)	100	98.0	97.1	92.5	93.1	96.2	96.9
AITL (N=15)	99.9	98.5	98.2	94.3	94.4	97.1	97.0

(b) The evaluation against 11 defense models

	$\text{Incv3}_{\text{ens3}}$	$\text{Incv3}_{\text{ens4}}$	$\text{IncResv2}_{\text{ens}}$	HGD	R&P	NIPS-r3	Bit-Red	JPEG	FD	ComDefend	RS
AITL (N=5)	69.9	65.8	43.4	50.4	46.9	59.9	51.6	87.1	73.0	83.2	39.5
AITL (N=10)	75.7	73.5	50.7	60.5	55.8	69.5	57.7	93.8	78.2	88.5	46.6
AITL (N=15)	80.4	78.0	54.8	66.1	58.6	72.0	60.8	95.0	80.8	91.0	47.3

ranging from 5 to 15. As shown in Tab. 4, more repetitions mean higher attack success rates, but it also brings about a higher computation cost. By increasing N from 5 to 10, the improvements of attack success rates on both normally trained models and defense models are significant. When further increasing N to 15, the improvement is slightly smaller, especially on the normally trained models. For a fair comparison of different methods, we control the number of repetitions per iteration in all methods to 5, *i.e.*, $m = 5$ in SIM [36], $m_2 = 5$ in Admix [62], $m = 5$ in AutoMA [75] and $N = 5$ in our AITL.

Table 5: The attack success rates under different number of image transformation operations in each combination, *i.e.*, M in Fig. 2 and Fig. 3. The adversarial examples are crafted on Incv3. * indicates the white-box model

(a) The evaluation against 7 normally trained models

	Incv3*	Incv4	IncResv2	Resv2-101	Resv2-152	PNASNet	NASNet
AITL (M=2)	99.3	93.6	91.5	87.2	87.4	91.8	91.9
AITL (M=3)	99.2	94.5	92.6	88.1	88.7	93.0	93.0
AITL (M=4)	99.8	95.8	94.1	88.8	90.1	94.1	94.0

(b) The evaluation against 11 defense models

	$\text{Incv3}_{\text{ens3}}$	$\text{Incv3}_{\text{ens4}}$	$\text{IncResv2}_{\text{ens}}$	HGD	R&P	NIPS-r3	Bit-Red	JPEG	FD	ComDefend	RS
AITL (M=2)	64.6	58.6	36.0	42.5	38.9	52.1	41.7	84.9	65.0	74.9	32.0
AITL (M=3)	67.8	63.8	38.9	48.2	43.4	56.6	45.7	86.0	67.4	80.6	36.8
AITL (M=4)	69.9	65.8	43.4	50.4	46.9	59.9	51.6	87.1	73.0	83.2	39.5

The effect of the number M. We also investigate the effect of the number M of image transformation operations in each combination on the attack success rates. The experimental results are shown in Tab. 5. We find that increasing M from 2 to 3 can bring an obvious improvement in the attack success rates, but the improvement is marginal when further increasing M to 4. Therefore, considering the computational efficiency, we set M to 4 in other experiments instead of further increasing M. Noting that the number of transformations used in the strongest baseline ADSCM is also 4 (*e.g.*, Admix, resize, crop and scale), which is the same as ours, formulating fair experimental comparisons.

C.2 More Results of Attack on the Single Model

We conduct the adversarial attack on more models under the setting of the single model in this section. We choose Incv4, IncResv2 and Resv2-101 as the white-box model to conduct the attack, respectively, and evaluate the generated adversarial examples against both normally trained models and defense models. The results are shown in Tab. 6, Tab. 7 and Tab. 8, respectively. From the results we can clearly conclude that, when choosing different models as the white-box model to conduct the attack, our proposed AITL consistently achieve higher attack success rates on various black-box models, especially on the defense models. It also shows that the optimal combinations of image transformations selected by the well-trained AITL specific to each image can successfully attack different models, *i.e.*, our AITL has a good generalization.

C.3 Attacks under Different Perturbation Budgets

We conduct the adversarial attacks under different perturbation budgets ϵ, ranging from 2 to 32. All experiments utilize Inceptionv3 [52] model as the white-box model. The curve of the attack success rates *vs.* different perturbation budgets is shown in Fig. 6. From the figure, we can clearly see that our AITL has the highest attack success rates under various perturbation budgets. Especially in the evaluation against the defense models, our AITL has an advantage of about 10% higher attack success rates over the strongest baseline ADSCM in the case of large perturbation budgets (*e.g.*, 16 and 32).

C.4 Visualization

The visualization of adversarial examples crafted on Incv3 by our proposed AITL is provided in Fig. 7.

C.5 Combined with Other Base Attack Methods

We combine our AITL with other base attack methods (*e.g.*, MIFGSM [15], TIM [16], NIM [36]) and compare the results with AutoMA [75]. As shown in Tab. 9, our method achieves significantly higher attack success rates than AutoMA [75] in all cases, both for normally trained and adversarially trained models, which clearly demonstrates the superiority of our AITL.

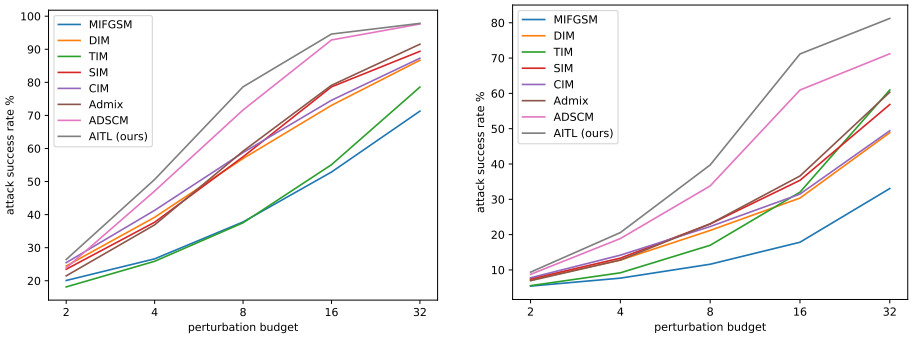
Table 6: Attack success rates (%) of adversarial attacks against 7 normally trained models and 11 defense models under **single model** setting. The adversarial examples are crafted on **Incv4**. * indicates the white-box model. † The results of AutoMA [75] are cited from their original paper

(a) The evaluation against 7 normally trained models

	Incv3	Incv4*	IncResv2	Resv2-101	Resv2-152	PNASNet	NASNet
MIFGSM [15]	65.9	**100**	55.0	37.3	38.0	48.6	46.6
DIM [71]	84.9	99.5	79.4	57.9	56.3	68.4	67.3
SIM [36]	87.2	99.9	81.9	69.4	69.0	72.8	75.4
CIM [74]	90.8	99.9	84.3	57.1	56.7	67.1	68.0
Admix [62]	81.9	99.9	76.3	63.9	61.4	68.3	69.6
ADSCM	92.6	99.8	89.5	77.1	78.2	82.2	82.7
Random	95.4	99.9	93.0	77.7	78.0	84.4	84.9
AITL (ours)	**97.0**	99.8	**95.3**	**87.8**	**88.9**	**92.4**	**93.7**
AutoMA-TIM† [68]	86.8	98.1	78.8	71.4	-	-	-
AITL-TIM (ours)	**94.7**	**99.8**	**92.8**	**89.4**	**89.4**	**91.7**	**92.4**

(b) The evaluation against 11 defense models

	$Incv3_{ens3}$	$Incv3_{ens4}$	$IncResv2_{ens}$	HGD	R&P	NIPS-r3	Bit-Red	JPEG	FD	ComDefend	RS
MIFGSM [15]	20.1	18.6	9.4	9.3	9.0	13.7	21.0	37.5	38.4	29.7	17.2
DIM [71]	36.1	34.1	18.9	25.9	21.6	30.7	28.1	60.8	47.8	49.0	23.8
SIM [36]	55.0	50.8	32.9	35.5	33.4	42.1	38.7	71.8	57.8	62.5	32.1
CIM [74]	37.5	33.9	19.9	29.9	21.2	30.4	29.0	62.6	46.8	47.4	23.3
Admix [62]	41.4	39.2	24.8	22.8	23.8	31.4	34.0	61.2	53.6	53.1	27.9
ADSCM	62.1	57.6	39.7	45.7	43.7	53.0	47.0	80.6	66.8	73.0	37.6
Random	56.8	50.1	33.8	40.3	37.1	48.0	40.9	81.4	59.8	71.7	32.8
AITL (ours)	**75.4**	**73.7**	**55.8**	**66.6**	**59.0**	**71.4**	**56.9**	**89.5**	**76.6**	**85.1**	**46.6**
AutoMA-TIM† [68]	76.0	75.5	67.4	69.6	68.0	71.0	-	-	82.1	-	-
AITL-TIM (ours)	**82.7**	**79.6**	**72.4**	**79.6**	**71.8**	**79.4**	**62.4**	**88.3**	**83.2**	**87.2**	**57.3**



(a) The average attack success rates against 7 normally trained models

(b) The average attack success rates against 11 defense models

Fig. 6: The curve of the attack success rates *vs.* different perturbation budgets

Table 7: Attack success rates (%) of adversarial attacks against 7 normally trained models and 11 defense models under **single model** setting. The adversarial examples are crafted on **IncResv2**. * indicates the white-box model. † The results of AutoMA [75] are cited from their original paper

(a) The evaluation against 7 normally trained models

	Incv3	Incv4	IncResv2*	Resv2-101	Resv2-152	PNASNet	NASNet
MIFGSM [15]	72.2	63.4	99.3	47.4	46.5	51.9	52.0
DIM [71]	85.3	82.7	98.5	66.8	67.2	68.0	72.3
SIM [36]	92.0	88.2	**99.8**	77.9	77.8	77.1	81.0
CIM [74]	90.6	88.4	99.4	69.4	68.3	70.8	73.9
Admix [62]	87.0	81.4	**99.8**	70.7	71.5	72.6	75.4
ADSCM	94.0	91.3	99.6	85.0	84.4	84.7	86.9
Random	94.8	91.3	99.2	84.9	84.5	86.3	87.8
AITL (ours)	**95.7**	**94.3**	99.1	**89.6**	**89.6**	**92.0**	**92.3**
AutoMA-TIM† [68]	87.2	85.6	98.3	79.7	-	-	-
AITL-TIM (ours)	**94.3**	**92.1**	**99.0**	**91.6**	**91.3**	**93.3**	**95.2**

(b) The evaluation against 11 defense models

	$Incv3_{ens3}$	$Incv3_{ens4}$	$IncResv2_{ens}$	HGD	R&P	NIPS-r3	Bit-Red	JPEG	FD	ComDefend	RS
MIFGSM [15]	28.8	23.0	15.0	17.1	14.2	18.1	25.0	48.5	43.9	38.7	18.8
DIM [71]	49.3	43.3	30.1	39.8	32.0	42.9	33.5	69.4	54.9	60.3	26.3
SIM [36]	63.3	55.1	48.5	47.3	41.2	51.8	43.8	80.6	62.5	70.9	33.3
CIM [74]	53.8	45.4	32.5	44.6	35.0	45.4	33.5	73.5	54.6	58.3	26.1
Admix [62]	51.6	44.3	35.1	32.2	30.2	39.9	39.6	70.5	58.1	61.7	30.1
ADSCM	70.4	65.4	51.7	56.9	55.1	63.1	53.8	83.1	72.3	77.5	40.1
Random	67.5	59.5	48.6	54.1	49.6	62.5	47.6	86.6	67.4	79.8	36.0
AITL (ours)	**79.9**	**74.6**	**65.4**	**71.5**	**67.5**	**75.8**	**62.3**	**89.8**	**79.8**	**86.1**	**49.4**
AutoMA-TIM† [68]	84.2	82.9	82.8	79.9	81.6	83.7	-	-	87.2	-	-
AITL-TIM (ours)	**88.6**	**85.0**	**85.4**	**83.4**	**84.2**	**86.2**	**71.7**	**89.4**	**88.4**	**90.4**	**65.4**

Table 8: Attack success rates (%) of adversarial attacks against 7 normally trained models and 11 defense models under **single model** setting. The adversarial examples are crafted on **Resv2-101**. * indicates the white-box model. † The results of AutoMA [75] are cited from their original paper

(a) The evaluation against 7 normally trained models

	Incv3	Incv4	IncResv2	Resv2-101*	Resv2-152	PNASNet	NASNet
MIFGSM [15]	50.5	40.6	41.9	98.9	86.5	61.4	60.6
DIM [71]	67.3	58.0	61.2	98.6	92.5	75.0	77.6
SIM [36]	59.4	49.8	53.4	99.5	93.8	75.3	76.4
CIM [74]	77.9	70.2	73.2	98.9	95.3	81.4	82.4
Admix [62]	54.2	45.4	46.3	**99.7**	90.2	71.7	70.4
ADSCM	73.9	63.3	68.5	99.5	94.9	83.3	84.0
Random	80.2	74.3	78.5	99.1	94.3	88.3	89.5
AITL (ours)	**81.5**	**78.3**	**79.4**	99.3	**96.1**	**91.2**	**91.3**
AutoMA-TIM† [68]	75.0	71.2	69.3	97.0	-	-	-
AITL-TIM (ours)	**79.5**	**75.4**	**73.8**	**98.3**	**96.2**	**89.2**	**89.5**

(b) The evaluation against 11 defense models

	$Incv3_{ens3}$	$Incv3_{ens4}$	$IncResv2_{ens}$	HGD	R&P	NIPS-r3	Bit-Red	JPEG	FD	ComDefend	RS
MIFGSM [15]	34.3	29.9	20.1	22.7	20.2	25.2	28.6	41.3	47.3	43.5	26.0
DIM [71]	52.6	45.4	33.6	35.9	33.4	41.4	37.3	61.5	57.9	62.0	35.9
SIM [36]	46.2	43.0	29.0	32.3	29.6	36.9	37.9	52.4	58.3	59.2	36.5
CIM [74]	64.7	60.2	44.1	51.6	45.9	56.5	44.5	73.0	62.7	70.5	42.6
Admix [62]	37.5	35.1	21.9	25.1	21.7	28.1	34.0	45.7	53.4	52.4	32.2
ADSCM	60.2	55.3	41.1	43.4	41.0	51.1	49.0	67.6	66.8	70.2	47.5
Random	66.5	62.7	47.3	52.1	48.1	60.1	52.4	75.2	70.8	78.6	47.7
AITL (ours)	**71.6**	**67.7**	**53.7**	**58.6**	**53.3**	**62.3**	**56.0**	**77.1**	**73.6**	**80.0**	**57.9**
AutoMA-TIM† [68]	73.4	74.6	68.2	67.1	67.6	71.7	-	-	82.3	-	-
AITL-TIM (ours)	**78.5**	**78.3**	**69.3**	**72.5**	**69.7**	**76.8**	**65.2**	**75.8**	**85.1**	**81.4**	**63.1**

Table 9: The comparison of AutoMA [75] and our AITL when combined with other base attack methods. The adversarial examples are crafted on Incv3. * indicates the white-box model. † The results of AutoMA [75] are cited from their original paper

	Incv3*	Incv4	IncResv2	Resv2-101	$Incv3_{ens3}$	$Incv3_{ens4}$	$IncResv2_{ens}$
MIFGSM [15]	**100**	52.2	50.6	37.4	15.6	15.2	6.4
AutoMA-MIFGSM† [75]	98.2	91.2	91.0	82.5	49.2	49.0	29.1
AITL-MIFGSM (ours)	99.8	**95.8**	**94.1**	**88.8**	**69.9**	**65.8**	**43.4**
TIM [16]	**99.9**	43.7	37.6	47.4	33.0	30.5	23.2
AutoMA-TIM† [75]	97.5	80.7	74.3	69.3	74.8	74.3	63.6
AITL-TIM (ours)	99.8	**93.4**	**92.1**	**91.9**	**81.3**	**78.9**	**69.1**
NIM [36]	**100**	79.0	76.1	64.8	38.1	36.6	19.2
AutoMA-NIM† [75]	98.8	88.4	86.4	80.2	41.5	39.3	21.8
AITL-NIM (ours)	**100**	**93.6**	**92.2**	**83.6**	**51.0**	**46.5**	**27.7**

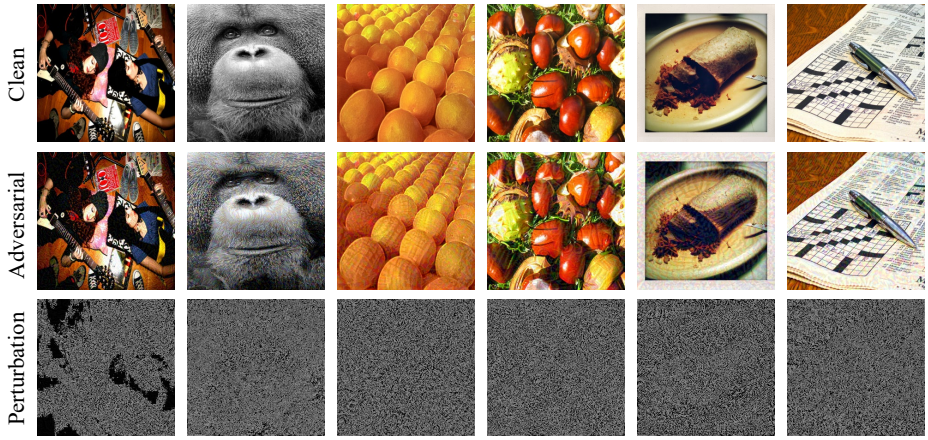


Fig. 7: Visualization of adversarial examples crafted on Incv3 by our proposed AITL

C.6 Compared with Ensemble-based Attack Methods

We also conduct experiments to compare our AITL with several ensemble-based attack methods. Liu *et al.* [**?**] first propose novel ensemble-based approaches to generating transferable adversarial examples. MIFGSM [15] studies three model fusion methods and finds that the method of fusing at the logits layer is the best. Li *et al.* [**?**] propose Ghost Networks to improve the transferability of adversarial examples by applying feature-level perturbations to an existing model to potentially create a huge set of diverse models. The experimental results of comparison between our AITL with these methods are shown in Tab. 10. The experimental setup is consistent with Tab. 3 in the manuscript, and our method achieves the best results among them.

Table 10: The comparison of our AITL and other ensemble-based attack methods

	Liu *et al.* [**?**]	Li *et al.* [**?**]	MIFGSM [15]	AITL(ours)
avg. of normally trained models	74.33	89.76	82.46	**97.36**
avg. of defense models	36.12	58.49	45.97	**84.57**

C.7 Compared with ATTA and Some Clarifications

The comparison between our AITL and ATTA [68] is presented in Tab. 11. The results of ATTA are directly quoted from their original paper, since the authors haven't released the code, and the details provided in the article are not enough to reproduce it. We also have tried to email the authors for more experimental details, but received no response.

Here we provide some explanations and clarifications for this comparison. We have some doubts about ATTA's experimental results, since the attack success rates against advanced defense models (*e.g.*, HGD, R&P, NIPS-r3 and so on) is significantly higher than that against normally trained models (*e.g.*, Incv4, IncResv2, Resv2-101), which is obviously counterintuitive. Their method seems to overfit the advanced defense models. Also, the results of some baseline methods (*e.g.*, MIFGSM, DIM, TIM) in their paper are quite different from other works (*e.g.*, NIM, AutoMA and ours). The inconsistency of experimental results may come from different experimental settings. We will do further comparisons with ATTA in the future.

Table 11: The comparison between ATTA [68] and our AITL. The adversarial examples are crafted on Incv3. * indicates the white-box model. † The results of ATTA [68] are cited from their original paper

(a) The comparison between ATTA and AITL

The evaluation against 7 models

	Incv3*	Incv4	IncResv2	Resv2-101	$\text{Incv3}_{\text{ens3}}$	$\text{Incv3}_{\text{ens4}}$	$\text{IncResv2}_{\text{ens}}$
ATTA† [68]	100	52.9	53.2	44.8	25.1	27.9	18.8
AITL (ours)	99.8	95.8	94.1	88.8	69.9	65.8	43.4

The evaluation against 6 advanced defense models

	HGD	R&P	NIPS-r3	FD	ComDefend	RS
ATTA† [68]	85.9	83.2	89.5	84.4	79.9	47.4
AITL (ours)	50.4	46.9	59.9	73.0	83.2	39.5

(b) The comparison between ATTA-TIM and AITL-TIM

	Incv3*	Incv4	IncResv2	Resv2-101	$\text{Incv3}_{\text{ens3}}$	$\text{Incv3}_{\text{ens4}}$	$\text{IncResv2}_{\text{ens}}$
ATTA-TIM† [68]	100	55.9	57.1	49.1	27.8	28.6	24.9
AITL-TIM (ours)	99.8	93.4	92.1	91.9	81.3	78.9	69.1

D Algorithms

The algorithm of training the Adaptive Image Transformation Learner is summarized in Algorithm 1.

The algorithm of generating the adversarial examples with pre-trained Adaptive Image Transformation Learner is summarized in Algorithm 2.

Algorithm 1 The training of Adaptive Image Transformation Learner

Input: the total training step T
Input: the learning rate β
Input: training set $(\mathcal{X}, \mathcal{Y})$, which represent the image and the corresponding label, respectively
Input: a source classifier model f, n target classifier models $f_1, f_2, \cdots, f_n$
Output: Transformation Encoder f_{en}, Transformation Decoder f_{de}, ASR Predictor f_{pre}, Feature Extractor f_{img} (denote their overall parameters as Θ)

1: **for** i $\in \{0, \cdots, T-1\}$ **do**
2: Get an image and corresponding label (x, y) from the dataset $(\mathcal{X}, \mathcal{Y})$
3: Extract the image feature from the feature extractor f_{img}

$$h_{img} = f_{img}(x)$$

4: Randomly sample M image transformation operations $t_1, t_2, \cdots, t_M$ as a combination, and represent them into one-hot codes $c_1, c_2, \cdots, c_M$
5: Embed one-hot codes into transformation features

$$a_1, a_2, \cdots, a_M = Embedding(c_1, c_2, \cdots, c_M)$$

6: Concatenate the transformation features into an integrated image transformation feature vector

$$a = Concat(a_1, a_2, \cdots, a_M)$$

7: Encode and decode the transformation feature vector

$$h_{trans} = f_{en}(a)$$
$$a' = f_{de}(h_{trans})$$

8: Reconstruct the image transformation operations

$$c'_1, c'_2, \cdots, c'_M = FC(a')$$

9: Predict the attack success rate

$$h_{mix} = Concat(h_{trans}, h_{img})$$
$$p_{asr} = f_{pre}(h_{mix})$$

10: Generate the adversarial examples x^{adv} by incorporating image transformation $t_1, t_2, \cdots, t_M$ into MIFGSM, *i.e.*, replacing Eq. (2) in MIFGSM by

$$g_{t+1} = \mu \cdot g_t + \frac{\nabla_{x_t^{adv}} \mathcal{J}(f(t_M \circ \cdots \circ t_1(x_t^{adv})), y)}{\|\nabla_{x_t^{adv}} \mathcal{J}(f(t_M \circ \cdots \circ t_1(x_t^{adv})), y)\|_1}$$

11: Calculate the corresponding actual average attack success rate q_{asr} by evaluating x^{adv} on $f_1, f_2, \cdots, f_n$

$$q_{asr} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}(f_i(x^{adv}) \neq y)$$

12: Calculate the loss function $\mathcal{L}_{total}$

13: Update the model parameter

$$\Theta = \Theta - \beta \cdot \nabla_{\Theta}\mathcal{L}_{total}$$

14: **end for**

Algorithm 2 Generating adversarial examples with AITL

Input: the original image x

Input: the number T of iteration steps during attack

Input: the number r of iterations during optimizing image transformation features

Input: the number N of repetitions of image transformation combination used in each attack step

Output: the adversarial example x_T^{adv}

1: Extract the image feature from the feature extractor f_{img}

$$h_{img} = f_{img}(x)$$

2: **for** i $\in \{0, \cdots, N-1\}$ **do**

3: Randomly sample M image transformation operations $t_1^i, t_2^i, \cdots, t_M^i$ as a combination, and represent them into one-hot codes $c_1^i, c_2^i, \cdots, c_M^i$

4: Embed one-hot codes into transformation features

$$a_1^i, a_2^i, \cdots, a_M^i = Embedding(c_1^i, c_2^i, \cdots, c_M^i)$$

5: Concatenate the transformation features into an integrated image transformation feature vector

$$a^i = Concat(a_1^i, a_2^i, \cdots, a_M^i)$$

6: Encode the transformation feature vector

$$h_{trans}^i = f_{en}(a^i)$$

7: Initialize the optimized transformation feature embedding

$$h_{trans}^{i,0} = h_{trans}^i$$

for j $\in \{0, \cdots, r-1\}$ **do**

Predict the attack success rate

$$h_{mix}^{j} = Concat(h_{trans}^{i,j}, h_{img})$$
$$p_{asr}^{j} = f_{pre}(h_{mix}^{j})$$

Update the transformation feature embedding

$$h_{trans}^{i,j+1} = h_{trans}^{i,j} + \gamma \cdot \nabla_{h_{trans}^{i,j}} p_{asr}^{j}$$

end for

Decode the transformation feature embedding

$$a_{opt}^{i} = f_{de}(h_{trans}^{i,r})$$

Reconstruct the one-hot image transformation vectors

$$\tilde{c}_1^i, \tilde{c}_1^i, \cdots, \tilde{c}_M^i = FC(a_{opt}^{i})$$

Achieve the corresponding image transformation operations $\tilde{t}_1^i, \tilde{t}_1^i, \cdots, \tilde{t}_M^i$

end for

$$x_0^{adv} = x, \quad g_0 = 0$$

for i $\in \{0, \cdots, T-1\}$ **do**

$$z = \frac{1}{N} \sum_{j=1}^{N} \frac{\nabla_{x_t^{adv}} \mathcal{J}(f(\tilde{t}_M^j \circ \cdots \circ \tilde{t}_1^j(x_t^{adv})), y)}{\|\nabla_{x_t^{adv}} \mathcal{J}(f(\tilde{t}_M^j \circ \cdots \circ \tilde{t}_1^j(x_t^{adv})), y)\|_1}$$
$$g_{t+1} = \mu \cdot g_t + z$$
$$x_{t+1}^{adv} = x_t^{adv} + \alpha \cdot sign(g_{t+1})$$

end for

return x_T^{adv}