

Adversarial Robustness Across Representation Spaces

Sruthy Annie Santhosh

Diego Coello de Portugal

Heliya Hasani

Aditya Nair

Supervised by: Mofassir ul Islam Arif

Contents

1. Introduction
2. Representation Spaces
3. Baseline
4. General Overview
5. Implementation
6. Results
7. Interpretation and Research Findings
8. Bibliography
9. Appendix

Introduction : Adversarial attacks

Adversarial attacks

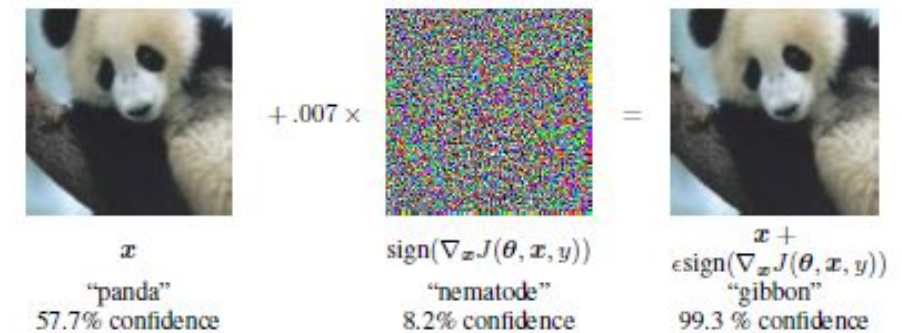
Generate perturbations on a data point to “fool” the model

Adversarial Robustness

Capability of the model to resist being “fooled”

Adversarial Learning

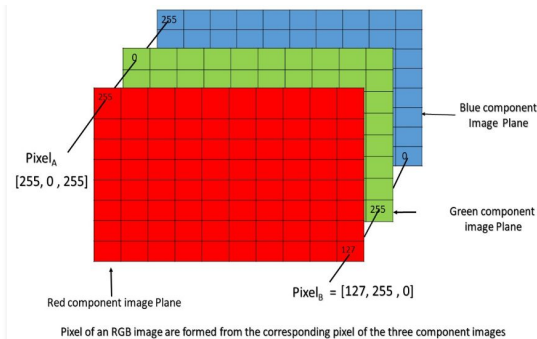
Training the model to increase its robustness using adversarial examples.



(Goodfellow, 2014)

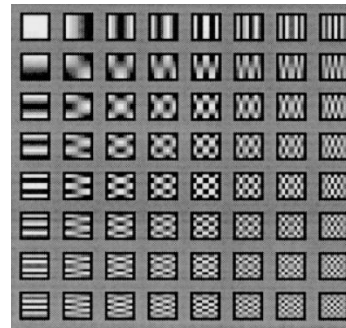
Introduction : What are representation spaces?

Pixel



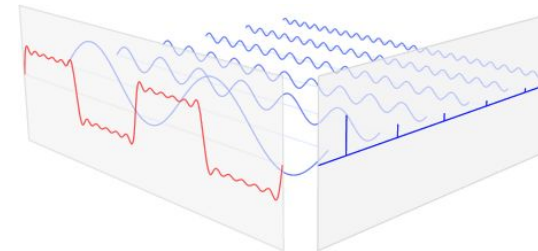
(Ihritik, 2018)

Discrete Cosine Transform (DCT)



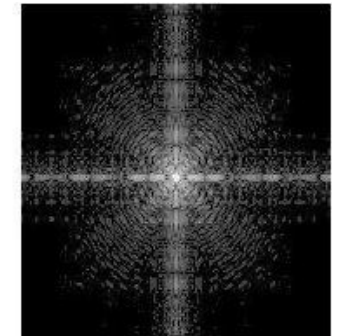
(Marshall, 2001)

Discrete Fourier Transform (DFT)



(Cao, 2013)

Log Transform



(Chaudhury, 2015)

Baseline - Adversarial Robustness Across Representation Spaces

- Importance of training against different attacks.
- **Representation spaces** → *Pixel, Discrete Cosine Transform (DCT)*
- Generate a list of attacks (type of attack and representation space).
- Select the attack and the representation space in a probabilistically way (*greedy, round robin or multiplicative weights method*).
- **Loss function** → *Cross entropy*

Baseline Assumption





	Test w/pixel ℓ_∞	Test w/DCT ℓ_∞	Nat. Acc.
Train w/pixel ℓ_∞	44.02 ± 1.02 	27.30 ± 1.44	80.24 ± 0.40 
Train w/DCT ℓ_∞	11.80 ± 0.68	51.92 ± 0.43 	74.92 ± 0.61 

Table 1: Shows the accuracies of **CIFAR10** dataset and is taken from the baseline paper (Yu, 2020)

	Test w/PGD_Pixel	Test w/PGD_DC T	Test w/PGD_FFT	Test Std(Nat). Acc
Train w/PGD_Pixel	52.79\pm0.45	38.64 \pm 1.0	68.83 \pm 0.35	73.45 \pm 0.83
Train w/PGD_DCT	50.53 \pm 1.02	49.86\pm0.86	65.92 \pm 1.64	69.23 \pm 1.73
Train w/PGD_FFT	14.48 \pm 1.39	5.83 \pm 0.51	77.88\pm0.43	86.06 \pm 0.46

Table 2: Results of our experiment on the **Fashion-MNIST** dataset robustness (\uparrow)

As can be seen no classifier is simultaneously robust to both types of attacks. This trend was observed for all the attacks (FGSM, FFGSM, PGD and PGD_L2)

General Overview After 2nd Presentation

	<u>2nd Presentation's Findings</u>	<u>Final Presentation's Findings</u>
Dataset	MNIST, Fashion-MNIST	CIFAR10
Baseline	Applied	Applied along with static and dynamic variations
Attacks	FGSM, PGD, FFGSM	PGD_L2
Training Methodology	Greedy, Round Robin, Multiplicative Weights Method	Holdout Testing also implemented with average calculation
Model	ResNet-50	ResNet-50
Representation Spaces	Pixel, DCT(+JPEG), DFT	Log Space

Current Implementation

- **The current model being used** → Resnet50 with CIFAR10 dataset
- **Attacks being used** → PGD, FGSM, FFGSM, PGD_L2
- **Representation spaces used** → Pixel, DCT (JPEG), DFT, LogSpace
- **Training methods implemented** → Greedy, Round Robin and Multiplicative Weights Update Method
- **Experiments conducted:**
 - Baseline implementation with 6 attacks
 - Selecting any 6 attacks from the list and training only on them (Holdout Testing)
 - Dynamically selecting 6 attacks and training on them in each epoch
 - Training the model on all the attacks and rep. spaces
- Standard accuracy values are also considered - on clean data
- Average accuracy also added

Methods Implemented

Methods Implemented	Description
Round Robin	Iterate over all the attacks recursively.
Greedy	Get loss for all the attacks before each epoch and train that epoch on the attack with highest loss
Multiplicative Weights	Store a weight for every attack and update it every epoch depending on the loss on that given attack. During the training epoch, select the attacks in a probabilistic way depending on the weights.

RESULT - FashionMNIST

Procedure	Std	Rand	FGSM	FGSM_DCT	FGSM_JPEG	FGSM_DFT	FFGSM	FFGSM_DCT	FFGSM_JPEG	FFGSM_DFT	PGD	PGD_DCT	PGD_JPEG	PGD_DFT
Std	88.29±0.47	61.5±6.38	5.28±1.5	13.26±2.56	10.56±3.1	33.1±3.14	10.12±2.23	7.75±1.65	7.47±1.68	72.03±0.78	0.0±0.0	0.0±0.0	0.0±0.0	23.26±4.08
Round Robin	80.37±1.61	79.71±1.96	31.76±1.27	57.76±1.54	49.1±1.88	68.03±1.47	70.26±1.71	68.11±1.73	69.2±1.82	78.33±1.69	42.79±1.65	32.92±1.85	36.54±2.04	72.35±1.99
Greedy	68.54±0.3	68.22±0.4	31.12±2.29	58.58±0.42	53.05±1.14	62.68±0.37	64.32±0.26	64.23±0.18	64.29±0.24	67.56±0.34	50.09±1.06	48.76±1.65	49.37±1.38	65.0±0.46
Mult. Weight	79.64±0.59	78.66±0.57	35.92±1.96	58.56±1.08	50.81±1.48	68.7±0.62	70.94±0.66	69.04±0.66	70.15±0.51	77.83±0.52	46.22±1.24	35.36±1.67	40.06±1.65	72.92±0.52

Adversarial Accuracy(↑) Obtained by Averaging 5 Times Over the Given Procedure

We can observe that, Multiplicative weights method performs the best over all attacks and representation spaces.

Green refers the best result.
Yellow refers to the second best result.

Results- CIFAR10

Reproducing Baseline Experiment

Procedures	Std (Nat. Acc.)	FGSM_ID	FGSM_DCT	PGD_ID	PGD_DCT	PGDL2_ID	PGDL2_DCT	AVG_ALL
Std	86.39+0.64	6.43+1.29	9.01+1.55	0.0+0.0	0.0+0.0	40.25+1.03	40.25+1.07	26.05+29.46
Round Robin	61.45+9.07	21.77+1.26	26.14+1.99	1.5+0.87	1.48+0.87	49.38+7.89	49.33+7.87	30.15+22.19
Greedy	32.47+4.73	21.08+1.67	25.08+3.06	19.68+1.25	21.79+2.34	30.79+4.41	30.79+4.42	25.95+4.94
Mult. Weight	62.9+15.43	21.97+3.26	26.76+4.76	3.32+2.72	3.61+3.13	49.38+11.52	49.34+11.48	31.04+21.74

Adversarial Accuracy(↑) Obtained by Averaging 5 Times Over the Given Procedure.

These results are comparable to those observed by the baseline paper. Multiplicative weights method performs the best overall.

CIFAR10- Baseline Variation

Training on a fixed set of 6 attacks (variation of baseline). Here we have trained on FGSM_FFT, FGSM_LOG, FFGSM_ID, PGD_FFT and PGDL2_DCT

Procedures	STD	RAND_ID	FGSM_ID	FGSM_FFT	FGSM_DCT	FGSM_JPEG	FGSM_LOG	FFGS_M_ID	FFGS_M_FFT	FFGS_M_DC_T	FFGS_M_JPEG	FFGS_M_LOG	PGD_ID	PGD_FFT	PGD_DCT	PGD_JPEG	PGD_LOG	PGDL2_ID	PGDL2_FFT	PGDL2_DCT	PGDL2_JPEG	PGDL2_LOG	AVG_ALL
Std	86.68 +0.42	81.75+ 0.5	11.35+ 3.07	24.96+ 1.59	12.69+ 2.98	11.82+ 2.94	84.72+ 0.43	8.29+2 .31	38.39+ 1.69	8.51+2 .37	8.08+2 .45	85.6+0 .36	3.78+3 .39	7.79+3 .73	3.74+3 .41	3.88+3 .49	84.69+ 0.43	37.21+ 1.45	21.34+ 1.82	37.21+ 1.47	37.23+ 1.46	86.51+ 0.44	35.74+ 32.13
Round Robin	86.04 +0.72	83.27+ 1.6	11.02+ 2.54	38.37+ 13.46	14.6+3 .08	13.39+ 2.84	84.74+ 0.36	14.39+ 6.44	51.46+ 13.14	15.75+ 7.55	16.0+8 .19	85.32+ 0.42	2.28+2 .88	21.07+ 13.57	2.23+2 .91	2.35+2 .97	84.72+ 0.36	52.7+1 5.53	19.36+ 2.6	52.71+ 15.54	52.71+ 15.53	85.94+ 0.67	40.47+ 31.5
Greedy	79.75 +8.91	77.81+ 7.84	15.88+ 7.2	44.73+ 14.21	21.48+ 10.05	20.66+ 10.54	78.81+ 8.41	23.02+ 13.29	54.79+ 11.72	25.51+ 15.12	26.0+1 5.64	79.23+ 8.62	6.14+5 .96	32.14+ 19.18	7.32+7 .59	8.75+9 .38	78.79+ 8.39	55.98+ 13.51	28.97+ 13.77	55.99+ 13.52	55.99+ 13.5	79.68+ 8.87	43.52+ 26.32
Mult. Weight	81.04 +8.03	79.32+ 7.28	15.52+ 6.31	45.88+ 12.5	20.94+ 8.79	19.9+9 .26	80.11+ 7.63	22.86+ 11.56	56.55+ 10.61	25.12+ 13.17	25.56+ 13.63	80.52+ 7.8	4.81+5 .65	31.53+ 16.71	5.65+7 .18	6.76+8 .83	80.1+7 .62	58.15+ 12.29	25.28+ 13.58	58.15+ 12.3	58.15+ 12.29	80.96+ 8.0	43.77+ 27.51

Adversarial Accuracy(↑) Obtained by Averaging 5 Times Over the Given Procedure

Here we can observe that, Multiplicative Weights outperforms other training procedures. This trend has been verified for different sets of attacks. Hence it can be found that the proposed method is not transferable.

CIFAR10- Baseline Variation

Using 6 attacks dynamically in each epoch for training (variation of baseline)

Procedures	Std (Nat. Acc.)	RAND_ID	FGSM_ID	FGSM_DCT	FGSM_JPEG	FGSM_FFT	FFGSM_ID	FFGSM_DCT	FFGSM_JPEG	FFGSM_FFT	PGD_ID	PGD_DCT	PGD_JPEG	PGD_FFT	PGDL2_ID	PGDL2_DCT	PGDL2_JPEG	PGDL2_FFT	AVG_AL
Std	86.9+0.2	80.71+2.11	7.84+1.26	10.31+1.52	8.46+1.26	25.76+1.1	6.52+1.17	7.99+1.26	6.64+1.31	39.04+0.7	0.01+0.02	0.0+0.01	0.0+0.0	1.95+0.71	40.72+0.93	40.8+0.94	40.87+0.94	4.53+0.74	22.73+26.21
Round Robin	77.96+4.1	76.72+4.14	18.18+3.34	25.84+3.03	24.48+3.28	54.65+2.87	29.44+2.58	33.46+2.95	33.59+2.58	63.58+3.39	2.04+1.34	2.35+1.77	2.93+1.9	42.52+3.53	66.23+3.21	66.2+3.25	66.24+3.3	36.78+3.32	40.18+24.7
Greedy	46.26+4.66	46.07+4.55	24.04+3.56	32.35+3.91	32.04+3.88	41.18+4.17	31.88+4.08	36.28+4.15	36.5+4.2	43.23+4.24	18.5+4.5	25.12+6.23	26.38+5.91	39.63+4.46	43.46+4.5	43.46+4.51	43.48+4.51	38.32+4.22	36.01+8.09
Mult. Weight	77.22+7.17	76.56+6.99	17.51+3.65	24.67+4.72	23.16+4.78	52.57+2.2	27.49+4.3	31.43+5.12	31.47+4.77	62.29+3.61	1.58+1.32	2.12+2.16	2.52+2.49	38.82+4.67	65.24+4.13	65.2+4.18	65.24+4.16	33.55+5.37	38.81+24.66

Adversarial Accuracy(↑) Obtained by Averaging 5 Times Over the Given Procedure.

Here we can observe that Round Robin performs the best.
Dynamically changing the attacks in each epoch, is inconsistent with the Baseline findings

CIFAR10

Training on all attacks and representation spaces

Procedures	Std (Nat. Acc.)	RAND_ID	FGSM_ID	FGSM_DCT	FGSM_JPEG	FGSM_FFT	FGSM_LOG	FFGS_M_ID	FFGS_M_DC_T	FFGS_M_JPEG	FFGS_M_FFT	FFGS_M_LOG	PGD_ID	PGD_DCT	PGD_JPEG	PGD_FFT	PGD_LOG	PGDL_2_ID	PGDL_2_DCT	PGDL_2_JPEG	PGDL_2_FFT	PGDL_2_LOG	AVG_ALL
Std	86.78 +0.26	82.04+ 1.55	6.95+1 .09	9.78+1 .45	7.81+1 .35	24.22+ 1.92	84.98+ 0.36	5.29+1 .05	6.58+1 .16	5.41+1 .14	37.95+ 1.49	85.78+ 0.29	0.0+0. 0	0.0+0. 0	0.0+0. 0	1.62+0 .36	84.94+ 0.37	39.94+ 0.95	39.98+ 0.88	39.94+ 0.9	4.27+0 .24	86.64+ 0.24	33.68+ 34.21
Round Robin	79.14 +1.84	77.89 +1.87	19.98 +1.27	27.7+ 1.44	26.11 +1.54	55.6+ 0.88	78.55 +1.74	30.5+ 1.53	34.68 +1.5	34.59 +1.48	64.78 +0.85	78.82 +1.75	1.4+0. 17	1.69+ 0.43	2.12+ 0.48	42.09 +1.33	78.55 +1.74	67.32 +1.01	67.31 +0.94	67.31 +1.01	36.29 +1.46	79.09 +1.81	47.8+ 27.0
Greedy	28.69 +2.59	28.68 +2.64	18.88 +1.52	22.34 +1.77	22.5+ 1.44	26.28 +1.98	28.57 +2.55	22.39 +0.96	23.97 +1.69	24.42 +1.57	27.09 +2.14	28.64 +2.59	16.39 +2.13	18.5+ 2.38	19.73 +1.79	25.42 +1.86	28.57 +2.55	27.16 +2.08	27.17 +2.1	27.18 +2.1	25.07 +1.69	28.67 +2.58	24.83 +3.71
Mult. Weight	77.64 +2.22	76.23 +2.87	19.56 +2.42	27.07 +1.95	25.38 +2.11	52.4+ 0.54	77.03 +2.18	28.6+ 1.63	32.53 +1.66	32.4+ 1.4	61.66 +1.02	77.31 +2.2	1.33+ 0.42	1.43+ 0.47	1.75+ 0.56	38.09 +0.99	77.02 +2.16	64.82 +1.08	64.84 +1.05	64.85 +1.13	34.55 +2.3	77.59 +2.22	46.09 +26.5

Adversarial Accuracy(↑) Obtained by Averaging 5 Times Over the Given Procedure

Training on all attacks and representation spaces does not improve the performance of the Multiplicative weights method. Round robin method outperforms the proposed method.

Research Findings

- Training in a specific representation space to increase robustness doesn't **translate** to other representation spaces.
- ***Multiplicative Weights*** method is better than Round Robin for small/medium amount of attacks/representation spaces (**<17**) in less complex data-sets (***MNIST, FashionMNIST***).
- However, the claim of ***Multiplicative Weights*** being **scalable** does not hold true for a higher amount of attacks (**>20**) and for more complex data-sets (***CIFAR10, CIFAR100***). Multiplicative Weights method tends to collapse, as more probability is given for attacks with low robustness. Therefore, there is no longer a significant difference between ***Multiplicative Weights*** and ***Round Robin*** training in this case.
- **Not** all representation spaces have the same impact for adversarial training. For example, **Log space** and **DFT** tend to squish down the perturbation after doing the inverse transformation, **reducing the impact** of the attack.

**THANK YOU FOR YOUR
INTEREST !!!**

Bibliography

- [1] Ian J. Goodfellow et al, *EXPLAINING AND HARNESSING ADVERSARIAL EXAMPLES*, 2014, arXiv:1412.6572
- [2] George Yu et al, *Adversarial Robustness Across Representation Spaces*, 2020, arXiv:2012.00802.
- [3] Wong, Eric and Rice, Leslie and Kolter, J Zico Fast is better than free: Revisiting adversarial training, 2020, arXiv preprint arXiv:2001.03994
- [4] Dave Marshall, *The Discrete Cosine Transform (DCT)*, 2001.
- [5] ihritik, *MATLAB | RGB image representation*, 2018, GeeksforGeeks.
- [6] Syed Ali Khayam, *The Discrete Cosine Transform (DCT): Theory and Application*, 2003, Michigan State University.
- [7] Shuhao Cao, Replicate the Fourier transform time-frequency domains correspondence illustration using TikZ, 2013, stackexchange
- [8] Saurabh Chaudhury , *Diffraction-Pattern-Enhanced-Through-Log-Transformation*, 2015

APPENDIX

- **Application made with Python**

- Python (Version = 3.7)

- **Libraries Used**

- Numpy (Version = 1.21.6)
- Matplotlib (Version = 3.2.2)
- Torch (Version = 1.7.1)
- Torchvision (Version = 0.8.2)
- Torchattacks (Version = 3.2.6)
- Torchjpeg

Appendix : Why use representation spaces?

What are representation spaces?

Structure of the elements in the dataset.

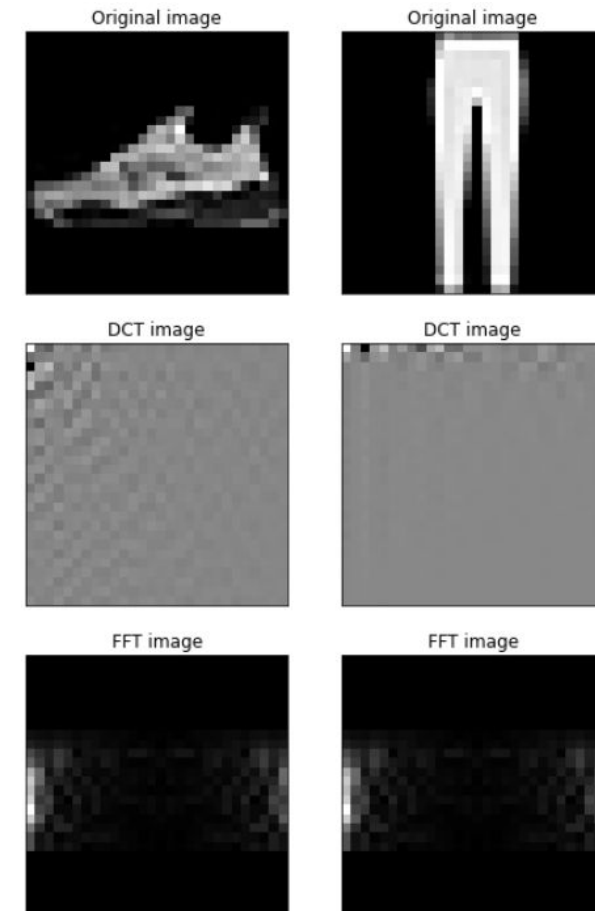
Eg → Pixel , DCT , DFT

Where are they used ?

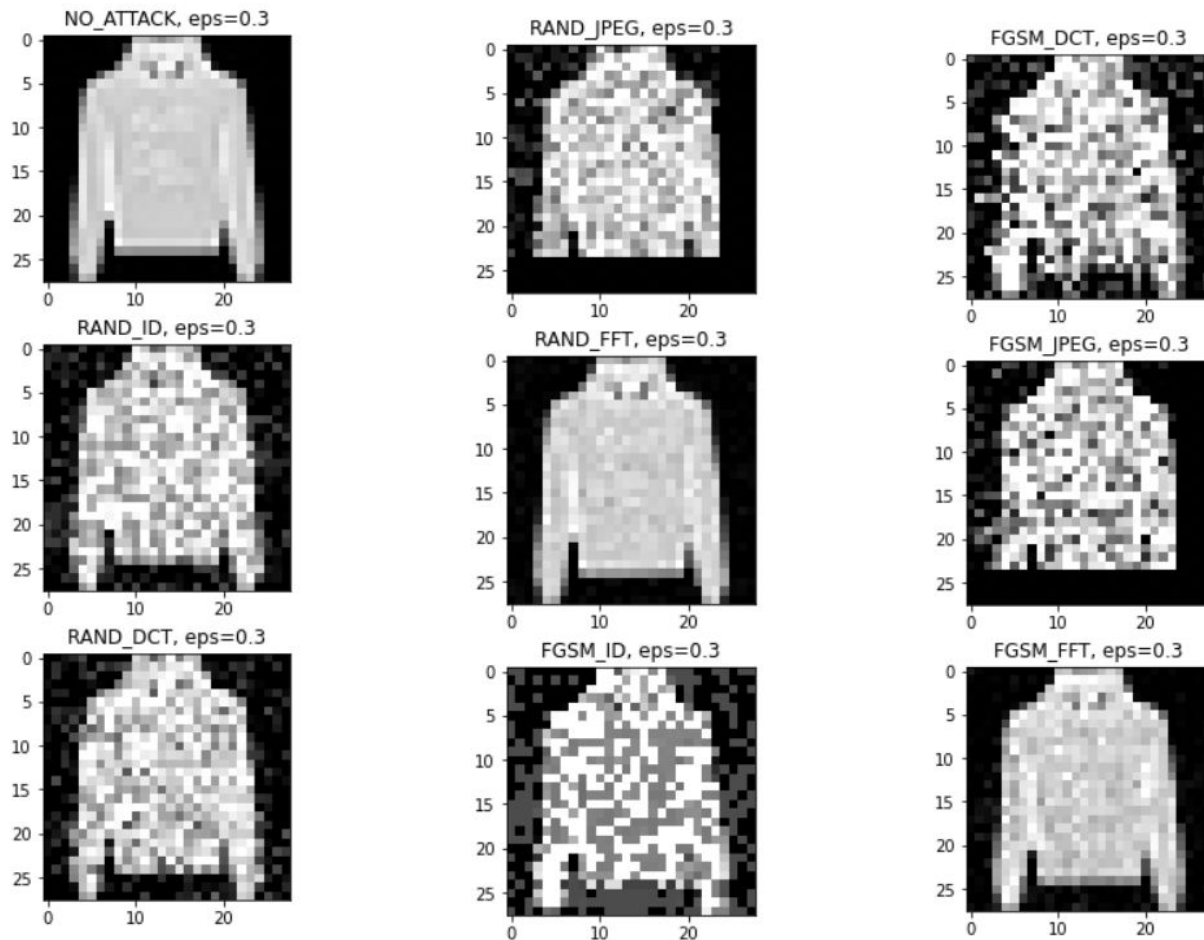
Pixels don't give precise information regarding to images. For more information waves (signals) should be considered. For instance, DCT is a type of a signal which is used in image compression because it is important that the pictures do not take up space on the computer. The primary reason is that we use Fourier series is that we can better analyze a signal in another domain rather in the original domain.

Why do we use them for adversarial attacks?

Adversarial robustness doesn't generalize well between different attacks or representation spaces.



Visualization - Images After Perturbation In Different Representation Spaces



Attacks Implemented

Attacks	Description	Norms
FGSM	It is a single step attack that uses the gradient of the loss w.r.t the input data, then adjusts the input data to maximize the loss using the sign of the gradient. It determines the gradient with regard to the image's pixels.	L_{inf}
FFGSM	FGSM combined with random initialization. It has lower computational requirement but similar performance.	L₂, L_{inf}
PGD	FGSM applied to an image iteratively. For L _{inf} and L2 norms, projection is linear-time in the dimension. The PGD attack seeks to locate a spot in the area that maximizes the loss.	L₂, L_{inf}
SLIDE	Sparse L1 Descent Attack, has finer control over the sparsity of an update step. Better suited than PGD for L ₁ norm.	L₁

APPENDIX

Mathematical expressions for attacks

FFGSM

In fast adversarial training (Wong, Rice, and Kolter 2020), a uniform randomization $\mathcal{U}(-\epsilon, \epsilon)$ is used instead of $\text{sgn}(\nabla_{x'} \ell(f(x'), y))$ in R+FGSM. Furthermore, for the first time, a step size α is set to a larger value than ϵ . Where $\Pi_{\mathcal{B}(x, \epsilon)}$ refers the projection to $\mathcal{B}(x, \epsilon)$. L_∞ is used as the distance measure.

$$x'_0 = x + \mathcal{U}(-\epsilon, \epsilon)$$

$$x' = \Pi_{\mathcal{B}(x, \epsilon)} \{x'_0 + \alpha \cdot \text{sgn}(\nabla_{x'_0} \ell(f(x'_0), y))\}$$

SLIDE

Sparse L1 Descent Attack, has finer control over the sparsity of an update step. Better than PGD for L_1 case because it suggest to use the top component of the gradient and the sign of those gradient.

Input: Input $x \in [0, 1]^d$, steps k , step-size γ , percentile q , ℓ_1 -bound ϵ
Output: $\hat{x} = x + r$ s.t. $\|r\|_1 \leq \epsilon$

```

r ← 0a
for 1 ≤ i ≤ k do
    g ← ∇rL(θ, x + r, y)
    ei = sign(gi) if |gi| ≥ Pq(|g|), else 0
    r ← r + γ · ei / ||e||1
    r ← ΠS1ε(r)
end

```

APPENDIX

Mathematical expressions for attacks

PGD

Before calculating the gradient a uniformly randomized noise is added to the original example. where $\Pi_{B(x, \epsilon)}$ refers the projection to $B(x, \epsilon)$ and $N(0_n, I_n)$ is a normal distribution. x_t denotes the adversarial $x_{t+1} = \Pi_{B(x, \epsilon)}\{x_t + \alpha \cdot \text{sgn}(\nabla_{x_t} \ell(f_\theta(x), f_\theta(x_t)))\}$ example after t -steps and α denotes a step size. L^∞ is used as the distance measure.

$$x'_0 = x + \mathcal{U}(-\epsilon, \epsilon)$$

$$x'_{t+1} = \Pi_{B(x, \epsilon)}\{x'_t + \alpha \cdot \text{sgn}(\nabla_{x'_t} \ell(f(x'_t), y))\}$$

FGSM

Here, J is the cost function used to train the neural network, θ represents the parameter of a model, x is the input to the model and y is the target associated with x (for machine learning tasks that have targets). Here, ϵ decides the size of the perturbation and sign of every element of the perturbation vector(or matrix/tensor) is decided by the sign of the input gradient at the element. This solution is motivated by linearizing the cost function and solving for the perturbation that maximizes the cost subject to an L^∞ constraint.

$$\eta = \epsilon \text{sign}(\nabla_x J(\theta, x, y))$$

Representation Spaces : DCT

DCT:

$$C(u, v) = \alpha(u)\alpha(v) \sum_{x=1}^{N-1} \sum_{y=1}^{N-1} f(x, y) \cos\left(\frac{\pi(2x+1)u}{2N}\right) \cos\left(\frac{\pi(2y+1)v}{2N}\right)$$

IDCT:

$$f(x, y) = \sum_{u=1}^{N-1} \sum_{v=1}^{N-1} \alpha(u)\alpha(v) C(u, v) \cos\left(\frac{\pi(2x+1)u}{2N}\right) \cos\left(\frac{\pi(2y+1)v}{2N}\right)$$

$$\alpha(u) = \begin{cases} \sqrt{\frac{1}{N}} & \text{if } x = 0 \\ \sqrt{\frac{2}{N}} & \text{if } x \neq 0 \end{cases}$$

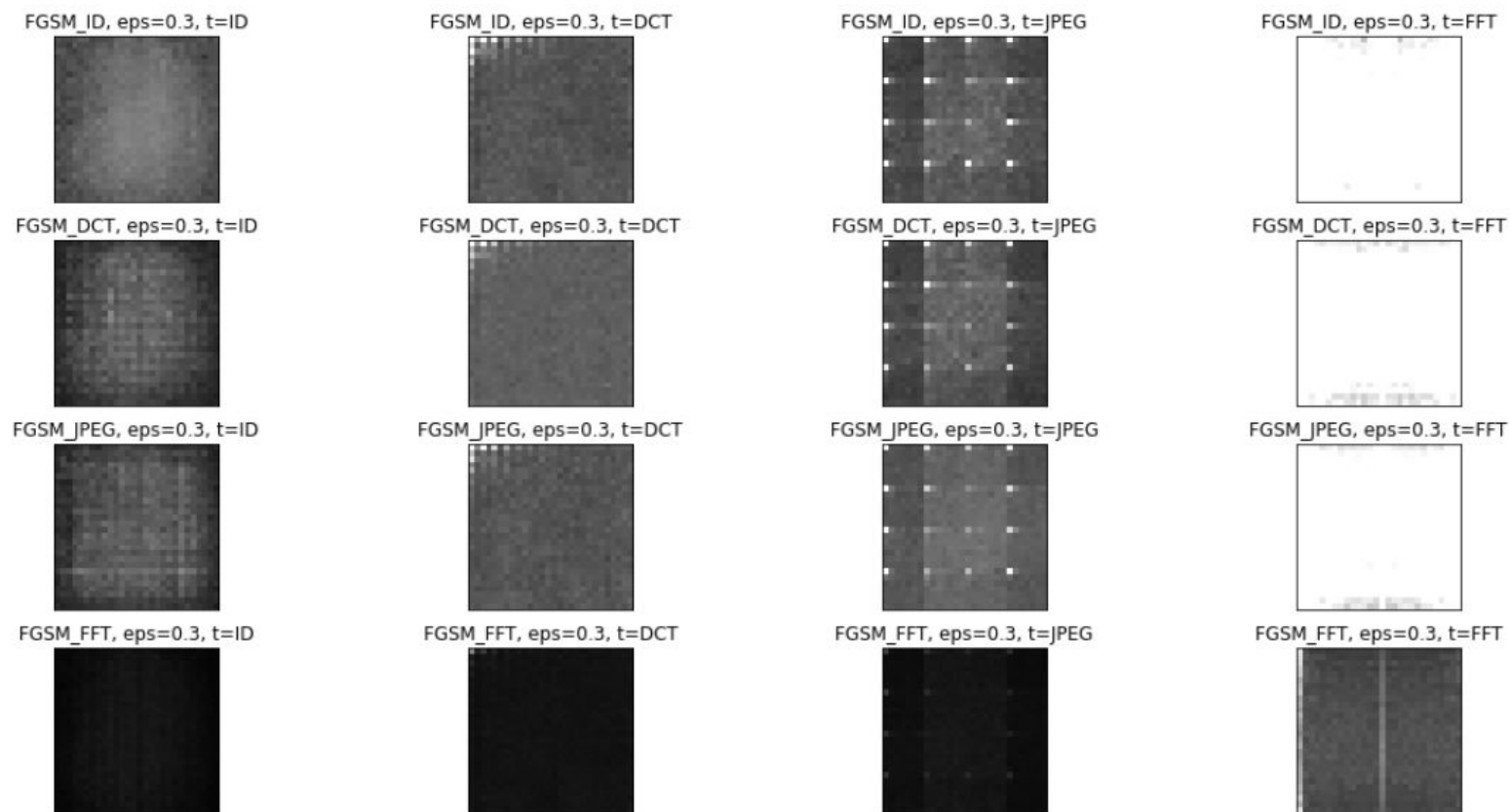
Representation Spaces : DFT

Fast Fourier Transform (FFT) is used to speed the space transformation process between *Pixel* representation and *Discrete Fourier Transform* representation

FFT:
$$F(u, v) = \frac{1}{N} \sum_{x=1}^{N-1} \sum_{y=1}^{N-1} f(x, y) e^{-2\pi i (\frac{ux}{N} + \frac{vy}{N})}$$

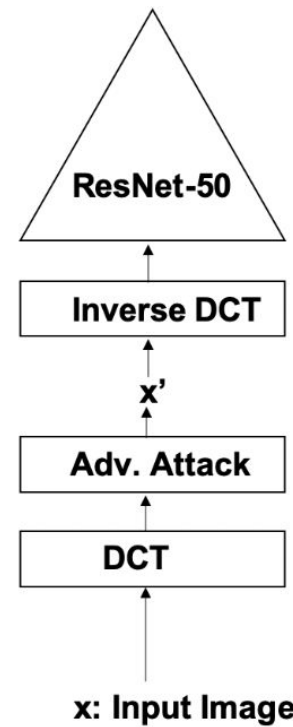
IFFT:
$$f(x, y) = \frac{1}{N} \sum_{u=1}^{N-1} \sum_{v=1}^{N-1} F(u, v) e^{2\pi i (\frac{ux}{N} + \frac{vy}{N})}$$

FGSM Perturbations Comparison



FGSM_FFT attack generates smaller perturbations (consistent across different representation spaces)

Flow Chart - Image Transformation



(Yu, 2020)

Figure 4. The modified network architecture for computing an adversarial perturbation in the DCT basis.

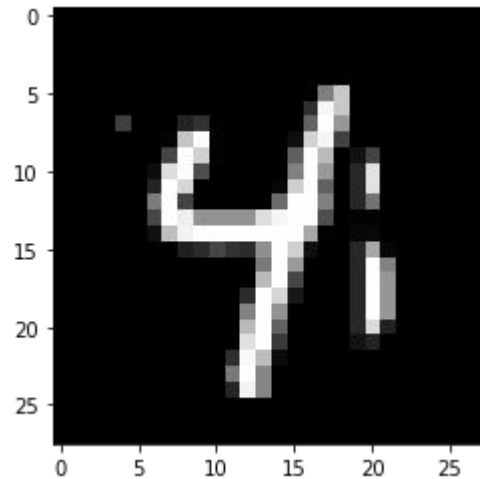
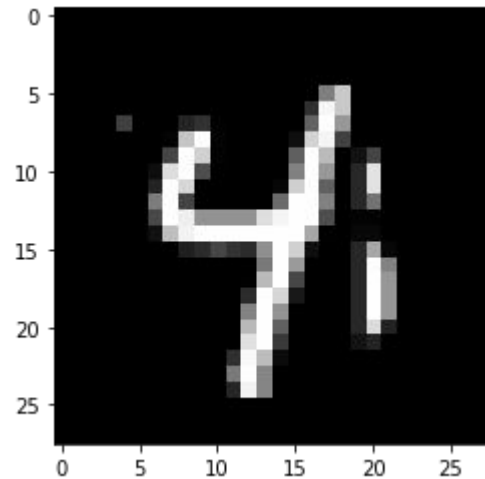
Algorithm: Multiplicative Weights Method

Input: Training data $\{(x_1, y_1), \dots, (x_m, y_m)\}$, Validation data $\{(x_{m+1}, y_{m+1}), \dots, (x_{m+s}, y_{m+s})\}$, mini batch size B , time steps T , update frequency r , window size h , Scaling factor η .

1. Initialize $w_i = 1$ for all $i \in [k]$.
2. For $t = 1 \dots, T$ do:
 - Repeat for r epochs:
 - Get the next mini batch of size B . Sample loss L_i with probability $p_i = \frac{w_i}{\sum_{j=1}^k w_j}$.
 - Run the PGD based algorithm to optimize L_i on the mini batch.
 - For all i set $w_i = w_i \cdot e^{\eta L_i^{\text{val}}(\theta_t)}$. Here L^{val} is the loss evaluated on the validation set.
3. Output $\hat{\theta} = \frac{1}{h} \sum_{t=T-h+1}^T \theta_t$.

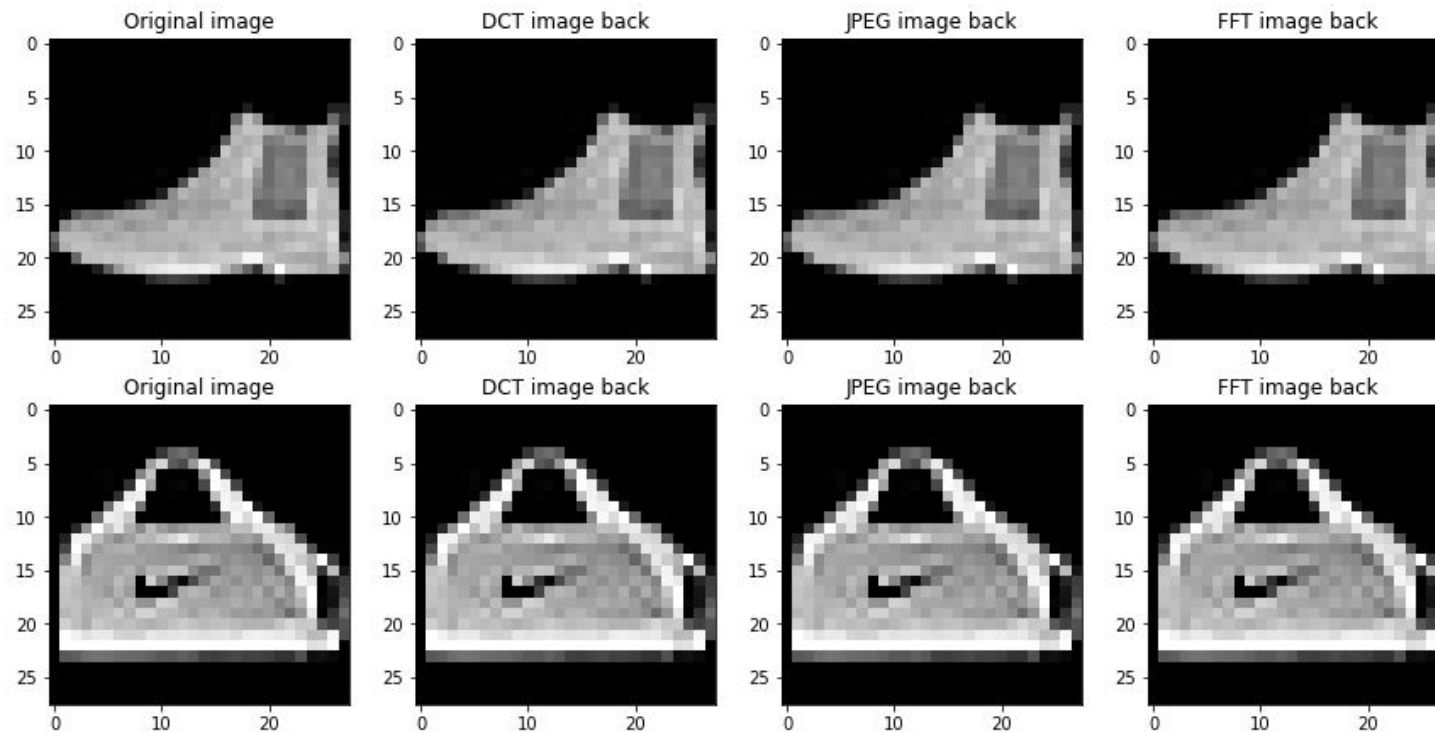
Result

MSE Loss between **Pixel** based images and **IDCT** converted images in ***MNIST dataset***



$$\underline{\text{Loss}} = 1.7110\text{e-}15$$

Result



Transformation between representation spaces causes only very small error.

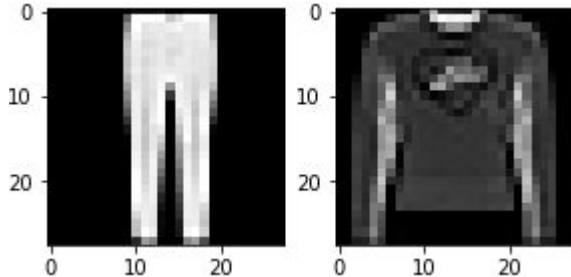
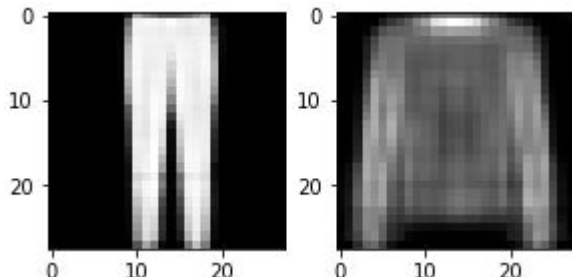
MSE loss after DCT transformation → $3.5082800922160706e-13$

MSE loss after JPEG transformation → $2.858904587133178e-14$

MSE loss after FFT transformation → $2.011194606444137e-15$

Reconstruction using AutoEncoders

An Auto-Encoder is used to reconstruct different sets of images to understand the impact of high frequency components in misclassification.

Sets of Inputs Used	Before(Auto-Encoder)	After(Auto-Encoder)
Original images in pixel	<p>images</p> 	<p>images</p> 

MSE loss incurred during **reconstruction** using AutoEncoder: 0.0001

Reconstruction Using AutoEncoders

We compared the loss between the different reconstructed images and it is found that adversarial images in pixel have the highest loss.

Sets of Inputs Used	MSLoss (Between Input and Reconstructed original pixel image)
Adversarial images in pixel (PGD)	0.01150194
Adversarial images in DCT (PGD)	0.01101580
Adversarial images in FFT (PGD)	0.00050876

We can see that the reconstruction error is insignificant.

Green refers the best result.
Orange refers to the second best result.

APPENDIX : FashionMNIST

More results with ffgsm training

	STD	Test w/FFGSM_Pixel	Test w/FFGSM_DCT	Test w/FFGSM_JPEG	Test w/FFGSM_FFT
STD	88.56+0.55	12.07+1.93	9.42+1.43	9.48+1.57	71.83+0.64
Train w/FFGSM_Pixel	83.96+0.27	74.99+0.48	71.1+0.39	73.54+0.58	81.99+0.27
Train w/FFGSM_DCT	83.41+0.35	73.93+0.09	72.19+0.29	73.11+0.11	81.63+0.36
Train w/FFGSM_JPEG	82.98+0.46	74.02+0.55	71.18+0.59	73.28+0.47	81.1+0.49
Train w/FFGSM_FFT	89.32+0.4	37.84+3.27	30.5+2.95	31.11+3.17	84.4+0.57

APPENDIX : FashionMNIST

More results with pgd training

	STD	Test w/PGD_Pixel	Test w/PGD_DCT	Test w/PGD_JPEG	Test w/PGD_FFT
STD	88.37+0.29	0.0+0.0	0.0+0.0	0.0+0.0	23.9+2.26
Train w/PGD_Pixel	71.22+1.97	52.79+0.45	38.64+1.0	46.73+1.2	68.83+0.35
Train w/PGD_DCT	71.27+3.1	50.53+1.02	49.86+0.86	50.22+0.9	65.92+1.64
Train w/PGD_JPEG	72.78+0.9	51.87+0.42	46.8+1.08	51.64+0.42	67.13+0.79
Train w/PGD_FFT	85.69+0.54	14.48+1.39	5.83+0.51	6.34+0.68	77.88+0.43

APPENDIX : CIFAR10

More results with fgsm training (JPEG)

	STD	Test w/FGSM_Pixel	Test w/FGSM_DCT	Test w/FGSM_JPEG	Test w/FGSM_FFT
STD	86.9+0.5	7.24+1.23	9.76+1.6	7.78+1.25	24.98+0.75
Train w/FGSM_Pixel	54.46+2.83	48.0+6.86	50.66+5.67	51.58+5.57	43.05+3.16
Train w/FGSM_DCT	45.08+5.87	20.26+1.94	71.91+2.94	37.97+6.5	32.73+3.11
Train w/FGSM_JPEG	60.77+0.58	20.54+0.24	41.22+0.2	40.14+0.26	54.16+0.46
Train w/FGSM_FFT	83.39+0.27	9.7+0.68	19.25+0.82	18.29+0.8	62.08+0.45

APPENDIX : CIFAR10

More results with ffgsm training (JPEG)

	STD	Test w/FFGSM_Pixel	Test w/FFGSM_DCT	Test w/FFGSM_JPEG	Test w/FFGSM_FFT
STD	86.64+0.38	5.22+0.97	6.64+1.24	5.33+1.12	38.57+1.08
Train w/FFGSM_Pixel	69.77+1.01	44.36+2.43	47.95+1.78	48.97+1.87	61.21+1.43
Train w/FFGSM_DCT	70.38+0.76	44.36+2.43	47.95+1.78	48.97+1.87	61.21+1.43
Train w/FFGSM_JPEG	71.0+0.74	34.44+0.36	49.46+0.41	49.55+0.46	64.78+0.56
Train w/FFGSM_FFT	86.41+0.24	16.8+0.87	21.1+0.68	21.82+0.85	67.71+0.48

APPENDIX : CIFAR10

More results with pgd training

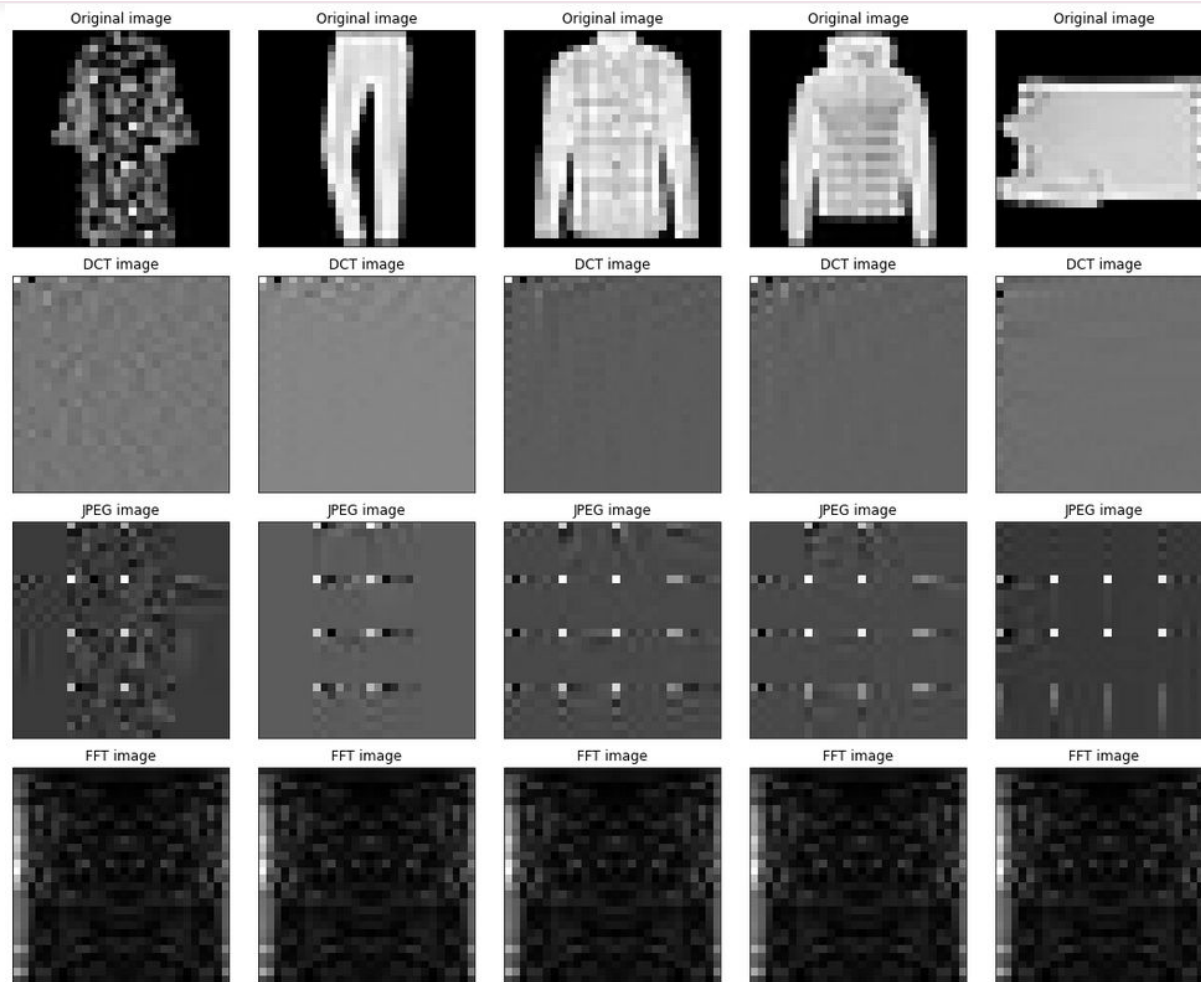
	STD	Test w/PGD_Pixel	Test w/PGD_DCT	Test w/PGD_JPEG	Test w/PGD_FFT
STD	86.12+0.56	0.0+0.0	0.0+0.0	0.0+0.01	1.71+0.43
Train w/PGD_Pixel	29.09+3.23	18.11+1.34	18.72+1.9	20.68+1.66	25.97+2.62
Train w/PGD_DCT	44.05+2.71	13.97+0.84	29.65+2.24	28.68+1.95	39.14+2.4
Train w/PGD_JPEG	43.61+4.38	15.73+0.56	29.0+2.94	29.2+2.87	38.96+4.05
Train w/PGD_FFT	80.22+0.38	1.6+0.19	4.34+0.24	5.78+0.21	58.11+0.07

APPENDIX : CIFAR10

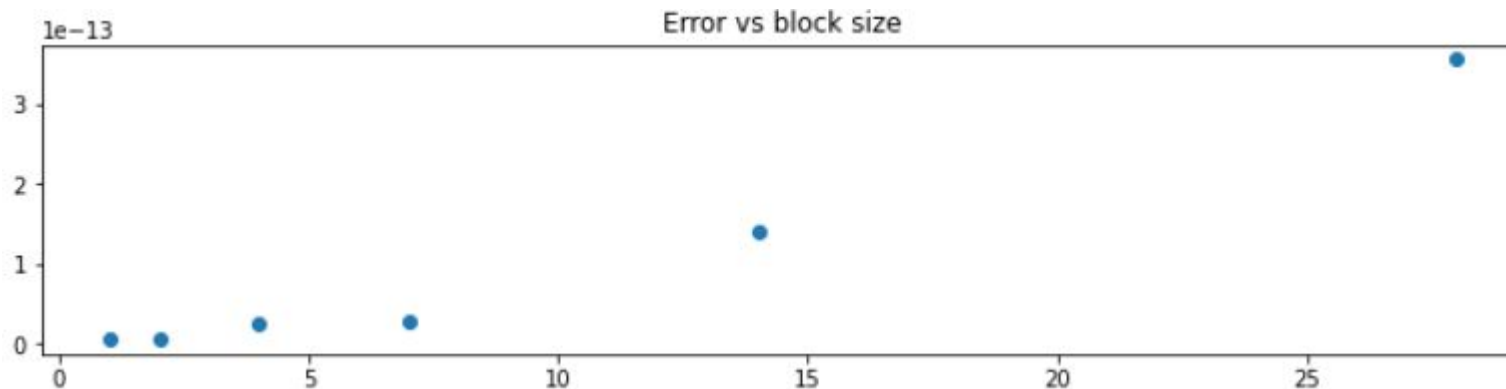
More results with pgdl2 training

	STD	Test w/PGDL2_Pixel	Test w/PGDL2_DCT	Test w/PGDL2_JPEG	Test w/PGDL2_FFT
STD	86.74+0.41	40.05+0.32	39.98+0.31	40.04+0.3	4.1+0.4
Train w/PGDL2_Pixel	87.66+0.24	69.08+0.42	69.09+0.42	69.05+0.39	17.87+0.57
Train w/PGDL2_DCT	87.38+0.15	69.01+0.21	69.03+0.17	69.06+0.22	17.8+0.52
Train w/PGDL2_JPEG	87.43+0.07	68.78+0.27	68.81+0.32	68.78+0.32	17.88+0.81
Train w/PGDL2_FFT	77.79+0.54	70.55+0.42	70.55+0.42	70.55+0.43	53.16+0.16

Visualization of representation spaces



JPEG block size comparison



MSE loss after **JPEG** transformation with block size **1**: **4.835885492772504e-15**

MSE loss after **JPEG** transformation with block size **2**: **5.475080363582553e-15**

MSE loss after **JPEG** transformation with block size **4**: 2.4232663944044608e-14

MSE loss after **JPEG** transformation with block size **7**: 2.7397946446836886e-14

MSE loss after **JPEG** transformation with block size **14**: 1.4177050646716621e-13

MSE loss after **JPEG** transformation with block size **28**: 3.573205099860477e-13

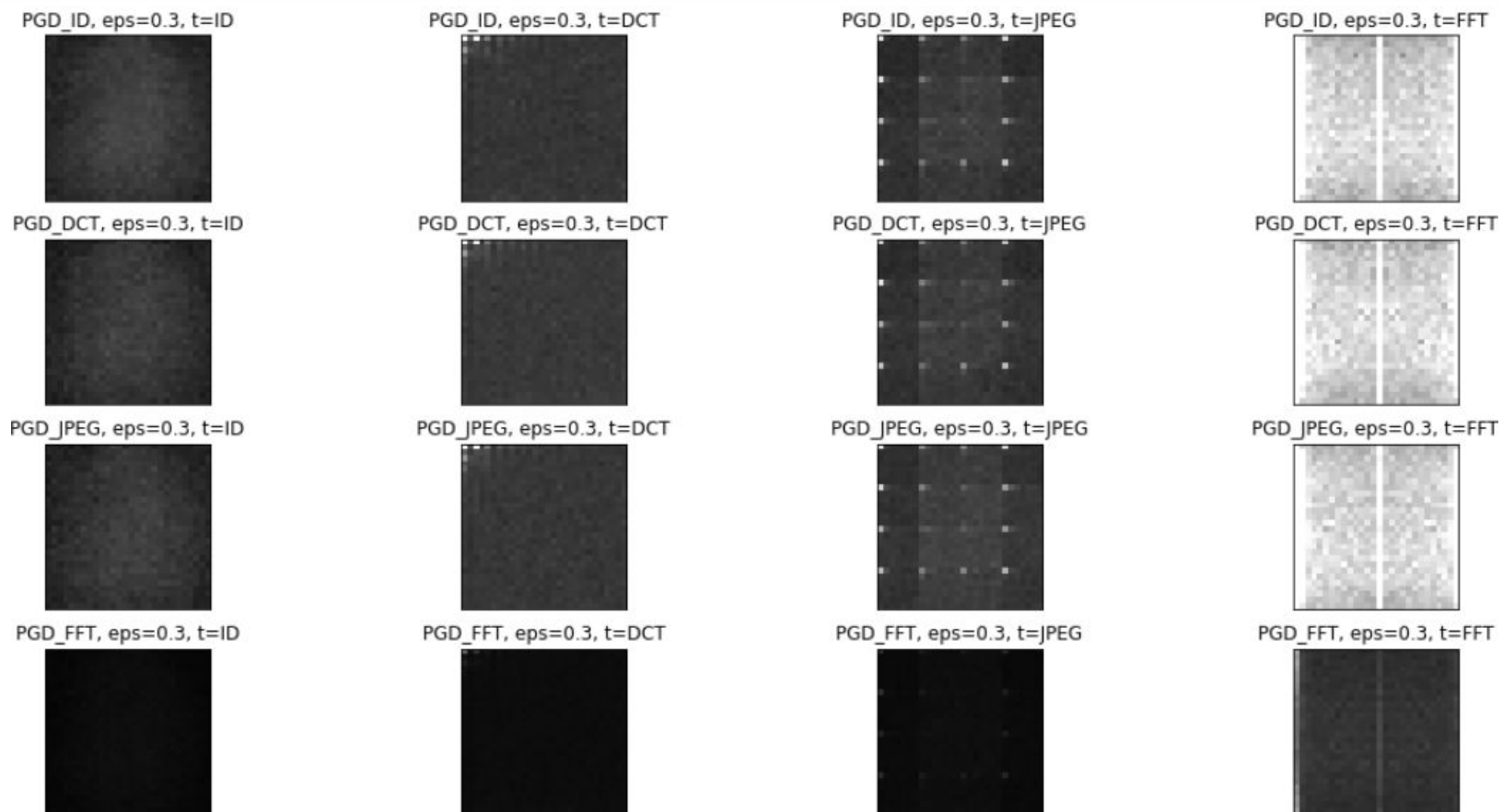
RESULT- FashionMNIST

Procedures	Std (Nat. Acc.)	Random	FGSM	PGD	PGD_DCT	PGD_DFT
Std	89.05±0.3	64.19±4.43	5.47±0.74	0.0±0.0	0.0±0.0	22.36±1.94
Round Robin	76.47±2.16	75.38±2.23	36.2±2.33	45.4±1.15	36.34±2.45	69.05±1.52
Greedy	69.63±1.16	69.29±1.15	30.06±2.63	49.78±1.3	48.2±1.55	65.7±1.26
Mult. Weight	77.2±1.94	76.71±1.73	34.31±3.67	45.12±1.08	34.81±1.97	69.45±1.78

Adversarial Accuracy(↑) Obtained by Averaging 5 Times Over the Given Procedure

Green refers the best result.
Orange refers to the second best result.

PGD Perturbations Comparison



PGD_FFT attack generates smaller perturbations (consistent across different representation spaces)

Appendix : Log transformation

```
def log_transform(data):  
    c = 255/np.log(2)  
    log_image = c*torch.log(1 + data)  
    return log_image
```

```
def inv_log(data):  
    c = 255/np.log(2)  
    inv = torch.exp(data/c)-1  
    return inv
```