

Dependable AI Report

# Minor Exam - 1

## Take Home Assignment

---

Ayush Abrol

B20AI052



## Question

**Aim:** Select a paper of your choice which is about attacking a deep learning model or system (adversarial attack, deepfake, or any backdoor attack) or defending against any of these attacks.

I decided the paper:

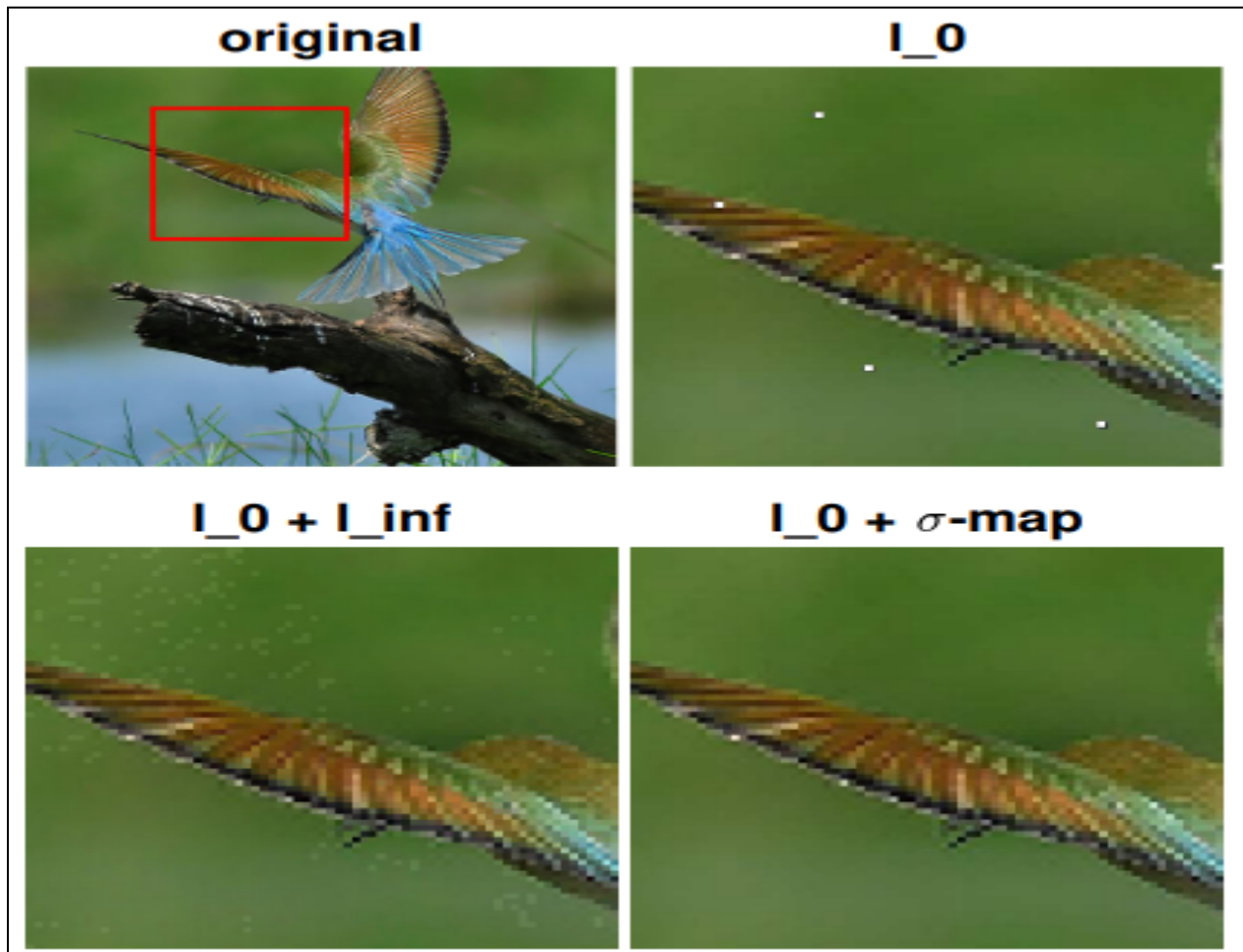
**Sparse and Imperceivable adversarial attacks:** [Link](#)

Tasks:

1. Reproduce the results of the research paper you have selected on one of the datasets used in the paper (train-test).
2. Show the results on a new dataset (not included in the paper) - fine-tune and test. Analyze the results.

**Abstract of the paper:** Numerous adversarial attacks have been shown to be capable of weakening neural networks. Highly sparse hostile attacks are particularly risky from a safety standpoint. On the other hand, sparse attacks generally result in massive pixelwise perturbations, making them potentially detectable. We provide a brand-new black-box method for creating adversarial examples that aims to reduce the  $L_0$ -distance from the source image. Extensive testing demonstrates that our attack is superior to or on par with the state of the art.

We may also integrate further restrictions on the componentwise perturbation. Our adversarial instances are virtually undetectable because we only allow pixel changes in regions of significant variance and refrain from allowing changes along axis-aligned edges. Additionally, we modify the Projected Gradient Descent attack to account for the componentwise integrating  $L_0$ -norm limitations. In order to strengthen the resilience of classifiers against sparse and undetectable adversarial alterations, this enables us to do adversarial training.



In this image,

**Upper Right:** This would be our  $I_0$ -attack, very few pixels, only 0.04%, are changed, but the modified pixels are clearly visible.

**Lower right:** This would be our sparse, 2.7% of the pixels are changed, but imperceptible attack ( $I_0 + \sigma$ -map).

### Procedure:

- All the necessary dependencies and libraries were taken care of initially like:
  - numpy, pandas, matplotlib for data manipulation and visualization.
  - **Torch**, torchvision, torch.nn, PIL for for implementing Deep Learning architectures where necessary using PyTorch.

- Set up my **environment** to **GPU** and cleared any already existing GPU cache because empty GPU memory is very necessary to completely implement the code for this research paper.
- Initially, I implemented some utility functions which would be helpful at multiple places such as:
  - **Calculating Logits.**
  - Function to return the **predictions** of the model.
  - Function to return the **predictions and the gradients** of the model using Cross Entropy Loss.
- Then, I implemented load\_dataset function which can load '**cifar10**', '**mnist**' and '**fashion\_mnist**' test datasets with n samples and their labels taken from the test set in the form of numpy arrays.
- Here, we are using **batch\_size = 1**, this is due to the GPU memory constraints.
- Then, built the **Basic ResNet (Residual Block) class with Batch Normalization and ReLU activation** where I specified the the number of in\_planes, planes and stride as the parameters to the class.
- Then, defined the class **Bottleneck ResNet Block (Residual Block) with Batch Normalization and ReLU activation** which looked like:

```
class Bottleneck(nn.Module):
    expansion = 4

    def __init__(self, in_planes, planes, stride=1):
        super(Bottleneck, self).__init__()
        self.conv1 = nn.Conv2d(in_planes, planes, kernel_size=1, bias=False)
        self.bn1 = nn.BatchNorm2d(planes)
        self.conv2 = nn.Conv2d(planes, planes, kernel_size=3, stride=stride,
                               padding=1, bias=False)
        self.bn2 = nn.BatchNorm2d(planes)
        self.conv3 = nn.Conv2d(planes, self.expansion * planes, kernel_size=1,
                               bias=False)
        self.bn3 = nn.BatchNorm2d(self.expansion * planes)

        self.shortcut = nn.Sequential()
        if stride != 1 or in_planes != self.expansion * planes:
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_planes, self.expansion * planes, kernel_size=1,
                           stride=stride, bias=False),
                nn.BatchNorm2d(self.expansion * planes)
            )

    def forward(self, x):
        out = F.relu(self.bn1(self.conv1(x)))
        out = F.relu(self.bn2(self.conv2(out)))
        out = self.bn3(self.conv3(out))
        out += self.shortcut(x)
        out = F.relu(out)
        return out
```

- Then, I implemented the **ResNet Model Class** which can be used to create ResNet models (BasicBlock or Bottleneck) of **different depths**.

- Then created the following different functions of ResNet Model:
  - ResNet18 (BasicBlock, [2, 2, 2, 2])
  - ResNet34 (BasicBlock, [3, 4, 6, 3])
  - ResNet50 (Bottleneck, [3, 4, 6, 3])
  - ResNet101 (Bottleneck, [3, 4, 23, 3])
  - ResNet152 (Bottleneck, [3, 8, 36, 3])
- Then, I defined the test function to test the model ResNet18 with a random test sample to check if its working.
- Now, we move on to creating our Attacks (**CornerSearch Attack and PGD attack** with modifications as mentioned in the paper).
- At, start we are defining functions for CornerSearch Attacks:
  - Function for **One-pixel modifications** to the image using the CornerSearch attack. This function returns a batch with the possible perturbations of the pixel in position position. In the first step we check all one pixel modifications of the original image  $x \in [0, 1]^{d \times 3}$  (color) or  $x \in [0, 1]^d$  (gray-scale). The tested modifications depend on the attack model.

1.  $l_0$ -**attack**: for each pixel  $i$  we generate  $8 = 2^3$  images changing the original color value to one of the 8 corners of the RGB color cube. Thus we name our method

**CornerSearch**. This results in a set of  $8d$  images, all one pixel modifications of the original image  $x$ , which we denote by  $(z^{(j)})_{j=1}^{8d}$ . For gray-scale images one just checks the extreme gray-scale values (black and white) and gets  $(z^{(j)})_{j=1}^{2d}$ .

2.  $l_0 + l_\infty$ -**attack**: for each pixel  $i$  we generate 8 images changing the original color value of  $(x_{ij})_{j=1}^3$  by the corners of the cube  $[-\epsilon, \epsilon]^3$  resulting again in  $(z^{(j)})_{j=1}^{8d}$  images. For gray-scale we use  $x_i \pm \epsilon$  resulting in total in  $(z^{(j)})_{j=1}^{2d}$  images. If necessary we clip to satisfy the constraint  $z^{(j)} \in [0, 1]^{d \times 3}$  or  $z^{(j)} \in [0, 1]^d$ .

3.  $l_0 + \sigma$ -**map attack**: for color images we generate for each pixel  $i$  two images by setting

$$y_{ij} = (1 \pm \kappa \sigma_{ij}) x_{ij}, \quad j = 1, \dots, 3,$$

where  $\kappa$  and  $\sigma_{ij}$  are as defined in Section 2. For gray scale images  $x \in [0, 1]^d$  we use

$$y_i = x_i \pm \kappa \sigma_i.$$

Finally, we clip  $y_{ij}$  and  $y_i$  to  $[0, 1]$ . Thus, this results in  $(z^{(j)})_{j=1}^{2d}$  images. We call it  $\sigma$ -**CornerSearch**.

- Function **flat2square** which returns the position and the perturbation given the index of an image of the batch of all the possible perturbations for all ‘L0’, ‘L0+Linf’ and ‘L0+sigma’ types of attack.
- Function for **npixels perturbation which** creates  $n\_iter$  images which differ from  $original\_x$  in at most  $k$  pixels for all ‘L0’, ‘L0+Linf’ and ‘L0+sigma’ types of attack.

**Multi-pixels modifications** Most of the times the modifications of one pixel are not sufficient to change the decision. Suppose we want to generate a candidate for a targeted adversarial sample towards class  $r$  by changing at most  $k$  pixels, choosing among the first  $N$  one-pixel perturbations according to the ordering  $\pi^{(r)}$ . We do this by sampling  $k$  indices  $(s_1, \dots, s_k)$  in  $\{1, \dots, N\}$  from the probability distribution on  $\{1, \dots, N\}$  defined as

$$P(Z = i) = \frac{2N - 2i + 1}{N^2}, \quad i = 1, \dots, N. \quad (6)$$

- Function for creating the **sigma-map** for a batch  $x$ . This map would be used in both **CornerSearch** and **PGD** attacks.
- Defining the **CornerSearch attack class** for all ‘L0’, ‘L0+Linf’ and ‘L0+sigma’ types of attack with the provided CornerSearch algorithm in the paper.

**Algorithm 1: CornerSearch**

**Input :**  $x$  original image classified as class  $c$ ,  $K$  number of classes,  $N$ ,  $k_{max}$ ,  $N_{iter}$   
**Output:**  $y$  adversarial example

```

1  $y \leftarrow \emptyset$ 
2 create one-pixels modifications  $(z^{(i)})_{i=1}^M$ 
3 if exists  $u \in (z^{(i)})_{i=1}^M$  classified not as  $c$  then
4    $y \leftarrow u$ , return
5 end
6 compute orderings  $\pi^{(1)}, \dots, \pi^{(K)}$ ,
7  $k \leftarrow 2$ 
8 while  $k \leq k_{max}$  do
9   for  $r = 1, \dots, K$  do
10    create the set  $Y^{(r)}$  of  $N_{iter}$  “ $k$ -pixels modifications” towards class  $r$  (see paragraph above)
11    if  $\exists u \in Y^{(r)}$  classified not as  $c$  then
12       $y \leftarrow u$ , return
13    end
14  end
15   $k \leftarrow k + 1$ 
16 end
```

- Now, we project our Corner Search algorithm onto the L0-ball and the L0+Linf ball using the projection algorithm provided in the paper.

We can write the projection problem onto  $C(x)$  as

$$\begin{aligned} \min_{z \in \mathbb{R}^{d \times 3}} \quad & \sum_{i=1}^d \sum_{j=1}^3 (y_{ij} - z_{ij})^2 \\ \text{s. th.} \quad & l_{ij} \leq z_{ij} \leq u_{ij}, \quad i = 1, \dots, d, j = 1, \dots, 3 \\ & \sum_{i=1}^d \max_{j=1,2,3} \mathbb{1}_{|z_{ij} - x_{ij}| > 0} > 0 \leq k \end{aligned}$$

Ignoring the combinatorial constraint, we first solve for each pixel  $i$  the problem

$$\begin{aligned} \min_{z_i \in \mathbb{R}^3} \quad & \sum_{j=1}^3 (y_{ij} - z_{ij})^2 \\ \text{s. th.} \quad & l_{ij} \leq z_{ij} \leq u_{ij}, \quad i = 1, \dots, d, j = 1, \dots, 3 \end{aligned}$$

The solution is given by  $z_{ij}^* = \max\{l_{ij}, \min\{y_{ij}, u_{ij}\}\}$ . We note that each pixel can be optimized independently from the other pixels. Thus we sort in decreasing order  $\pi$  the gains

$$\phi_i := \sum_{j=1}^3 (y_{ij} - x_{ij})^2 - \sum_{j=1}^3 (y_{ij} - z_{ij}^*)^2.$$

achieved by each pixel  $i$ . Thus the final solution differs from  $x$  in the  $k$  pixels (or less if there are less than  $k$  pixels with positive  $\phi_i$ ) which have the largest gain and is given by

$$z_{\pi, j} = \begin{cases} z_{\pi, j}^* & \text{for } i = 1, \dots, k, j = 1, \dots, 3, \\ x_{\pi, j} & \text{else.} \end{cases}$$

Using  $l_{ij} = 0$  and  $u_{ij} = 1$  we recover the projection onto the intersection of  $l_0$ -ball and  $[0, 1]^{d \times 3}$ . For  $l_0 + l_\infty$  note that the two constraints

$$0 \leq z_{ij} \leq 1, \quad -\epsilon \leq z_{ij} - x_{ij} \leq \epsilon,$$

are equivalent to:

$$\max\{0, -\epsilon + x_{ij}\} \leq z_{ij} \leq \min\{1, x_{ij} + \epsilon\}.$$

Thus by using

$$l_{ij} = \max\{0, -\epsilon + x_{ij}\}, \quad u_{ij} = \min\{1, x_{ij} + \epsilon\},$$

the set  $C(x)$  is equal to the intersection of the  $l_0$ -ball of radius  $k$ , the  $l_\infty$ -ball of radius  $\epsilon$  around  $x$  and  $[0, 1]^{d \times 3}$ .

$$C(x) = \left\{ z \in \mathbb{R}^{d \times 3} \mid \sum_{i=1}^d \max_{j=1,2,3} \mathbb{1}_{|z_{ij} - x_{ij}| > 0} \leq k, \right. \\ \left. l_{ij} \leq z_{ij} \leq u_{ij} \right\}.$$

- PGD (Projected Gradient Descent) Attacks
- Function **project\_L0\_box** - Projection of the batch y to a batch x such that:
  - Each image of the batch x has at most k pixels with non-zero channels
  - lb <= x <= ub
- Function **project\_L0\_sigma** - Projection of the batch y to a batch x such that:
  - 1. 0 <= x <= 1
  - Each image of the batch x differs from the corresponding one of x\_nat in at most k pixels.
  - (1 - kappa\*sigma)\*x\_nat <= x <= (1 + kappa\*sigma)\*x\_nat.

- Function **perturb\_L0\_box** - PGD attack wrt L0-norm + box constraints. It returns adversarial examples (if found) `adv` for the images `x_nat`, with correct labels `y_nat`, such that:
  - Each image of the batch `adv` differs from the corresponding one of `x_nat` in at most `k` pixels.
  - $lb \leq adv - x_{nat} \leq ub$
  - It returns also a vector of flags where 1 means no adversarial example found (in this case the original image is returned in `adv`).
- Function **perturb\_L0\_sigma** - PGD attack wrt L0-norm + sigma-map constraints. It returns adversarial examples (if found) `adv` for the images `x_nat`, with correct labels `y_nat`, such that:
  - Each image of the batch `adv` differs from the corresponding one of `x_nat` in at most `k` pixels.
  - $(1 - \kappa \cdot \sigma) \cdot x_{nat} \leq adv \leq (1 + \kappa \cdot \sigma) \cdot x_{nat}$ .
  - It returns also a vector of flags where 1 means no adversarial example found (in this case the original image is returned in `adv`)
- Then, finally defined the class PGD attack which would be used to create the PGD attack using L0-norm + box constraints, L0-norm + L<sub>inf</sub> constraints and L0-norm + sigma-map constraints.
- Running attacks on the models with CIFAR-10 dataset and evaluating the results.
- Note: In the research paper itself, the authors have used the CIFAR-100 dataset for their experiments with PyTorch implementation of the code. Along with that, the authors have used CIFAR-10 and MNIST datasets for their experiments with Tensorflow implementation of the code.
- Therefore, I will be using CIFAR-10 dataset for my experiments and reproducing the results of the paper with CIFAR-10 dataset.
- Later on, I will also try to reproduce the results of the paper with FashionMNIST dataset.
- We evaluate the effectiveness of our score-based I0-attack CornerSearch and our whitebox attack PGD. Moreover, we give illustrative examples of our sparse and imperceivable I0 +  $\sigma$ -map attacks  $\sigma$ -CornerSearch and  $\sigma$ -PGD.
- Loaded the test set for CIFAR-10 dataset for `n = 100` samples due to GPU constraints.

```
X_test shape: (100, 32, 32, 3)
y_test shape: (100,)
```



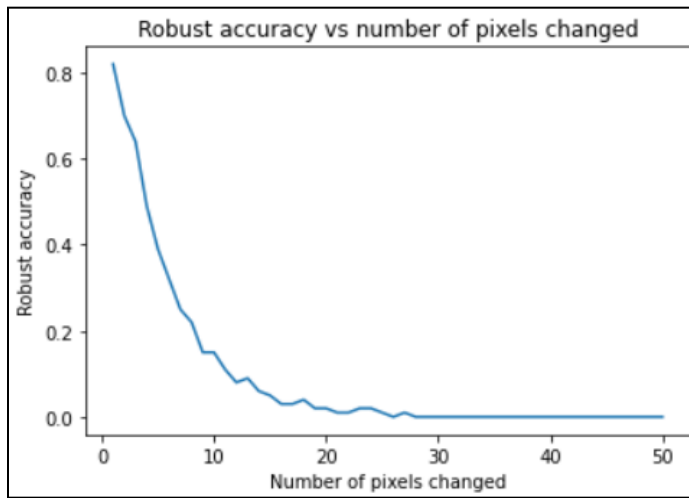
- Then, defined the ResNet18() object and loaded the parameters from the already trained CIFAR-10 dataset model which was trained on ResNet18 itself and is stored in the models directory named “model\_test.pt”
- Then, set the model for evaluation.
- Performed PGD attack with L0 norm with sparsity set to 5 pixels for 5 restarts.

```
Restart 1 - Robust accuracy: 0.36
Restart 2 - Robust accuracy: 0.3
Restart 3 - Robust accuracy: 0.26
Restart 4 - Robust accuracy: 0.24
Restart 5 - Robust accuracy: 0.23
Pixels changed: [5 5 5 5 5 5 5 5 5 5 5 0 5 0 0 5 5 5 5 0 5 5 5 0
5 5 5 0 5 5 5 5 5 5 0 5 5
5 0 5 5 5 5 5 0 0 5 5 0 5 0 5 5 5 5 5 0 5 5 5 0 5 5 5 5 5
5 5 5 5 5
5 0 5 5 5 5 5 0 0 0 5 5 5 5 0 0 0 5 5 5 5 5 5 0 0 5]
Robust accuracy at 5 pixels: 23.00%
Maximum perturbation size: 1.00000
```

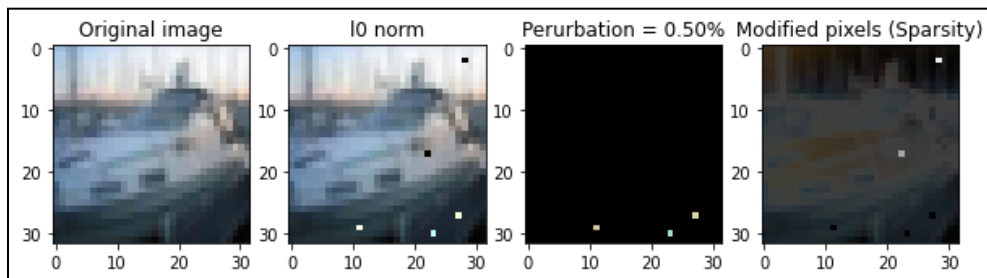
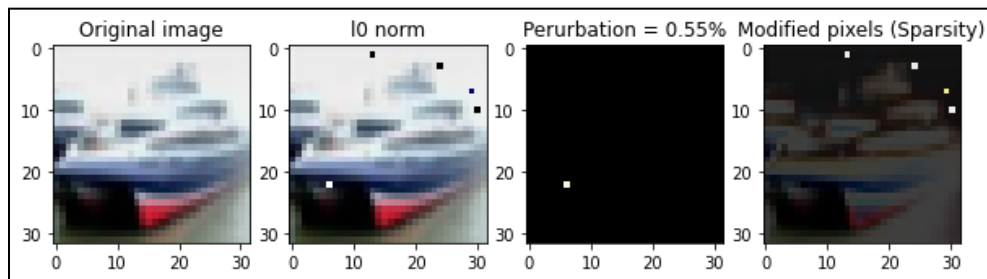
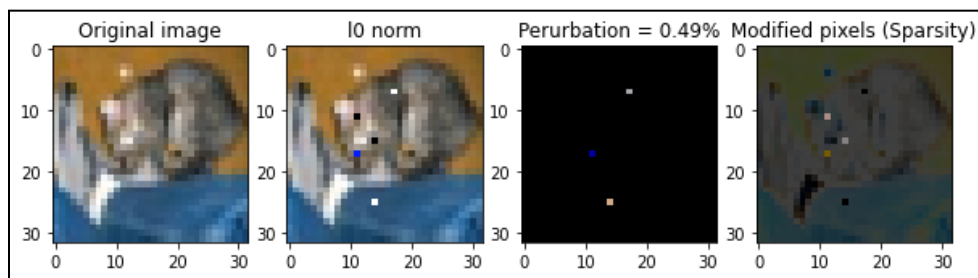
- Performed PGD attack with L0 norm with sparsity set to 10 pixels for 5 restarts.

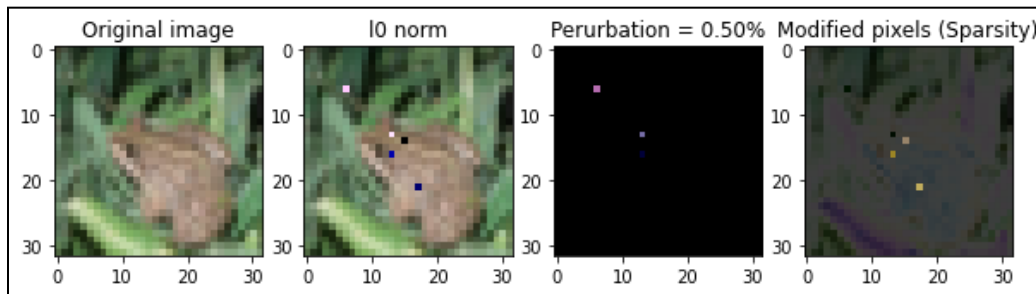
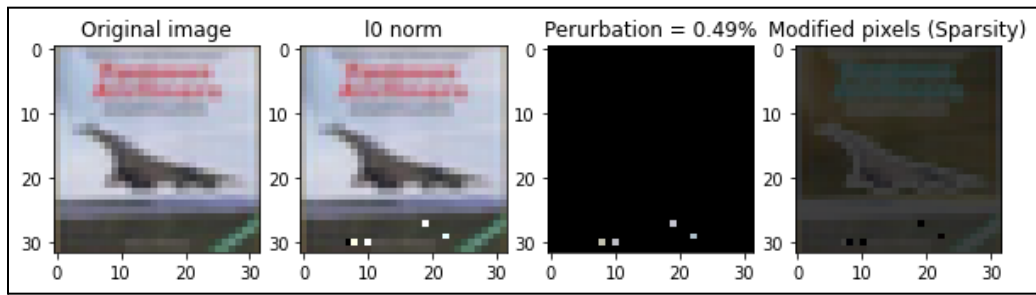
```
Restart 1 - Robust accuracy: 0.12
Restart 2 - Robust accuracy: 0.07
Restart 3 - Robust accuracy: 0.06
Restart 4 - Robust accuracy: 0.05
Restart 5 - Robust accuracy: 0.05
Pixels changed: [10 10 10 10 10 10 10 10 10 10 10 10 0 10 10 10 10
10 10 10 10 10 10 10 0
10 10 10 10 10 10 10 10 10 10 10 10 10 10 0 10 10 10 10 10 10
0 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 0 10 10
10 10 10
10 10 10 10]
Robust accuracy at 10 pixels: 5.00%
Maximum perturbation size: 1.00000
```

- Performed PGD attack with L0 norm with changing pixels from 1-50 with n-restarts = 1 and got the following robust accuracy vs number of pixels changed plot.



- Visualized images with PGD attack with l0 norm for sparsity set to 5 pixels.





- Performed PGD attack with L0+Linf norm with sparsity = 5 pixels for 5 restarts.

```
Restart 1 - Robust accuracy: 0.76
Restart 2 - Robust accuracy: 0.76
Restart 3 - Robust accuracy: 0.76
Restart 4 - Robust accuracy: 0.74
Restart 5 - Robust accuracy: 0.74
Pixels changed: [0 0 5 5 0 0 0 5 0 0 0 0 0 5 0 0 0 0 5 0 5 0
0 0 5 0 0 0 0 0 5 0 0 5 0
5 0 0 0 0 5 0 0 0 5 5 0 0 0 0 5 5 0 0 0 5 5 0 5 0 0 0 0 0 0
0 5 0 0 0
0 0 0 0 5 0 0 0 0 0 5 5 0 5 0 0 0 0 0 0 0 5 0 5 0 0]
Robust accuracy at 5 pixels: 74.00%
Maximum perturbation size: 0.10000
```

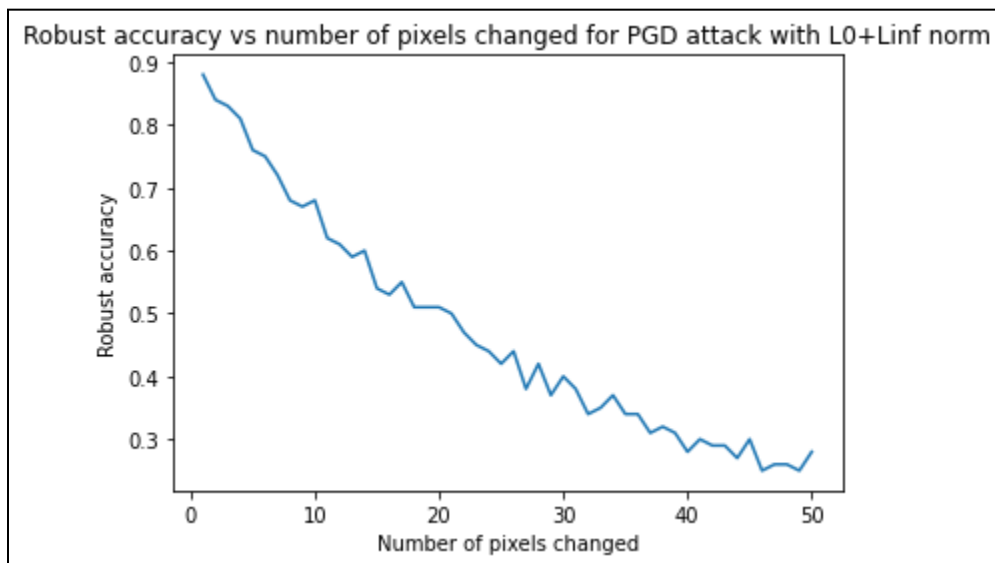
- Performed PGD attack with L0+Linf norm with sparsity = 10 pixels for 5 restarts.

```

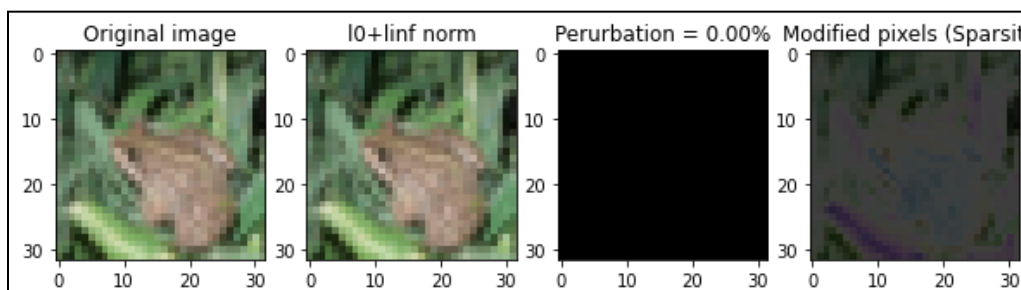
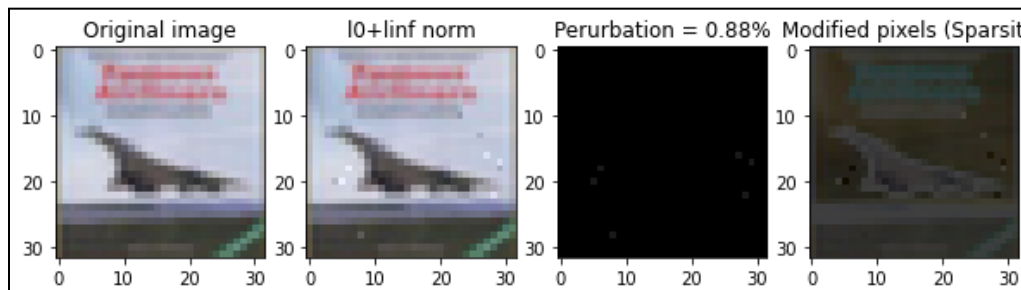
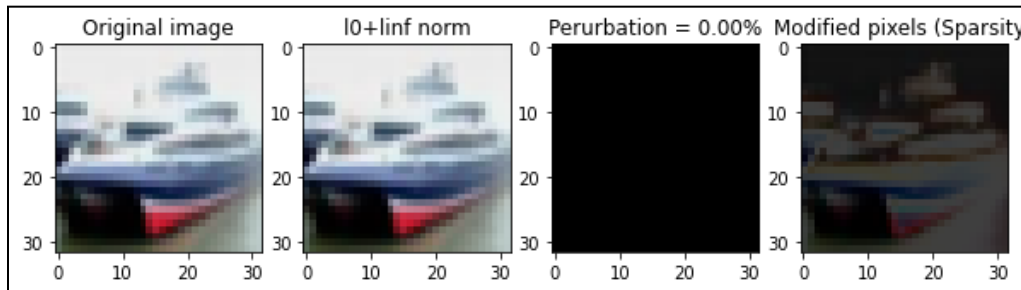
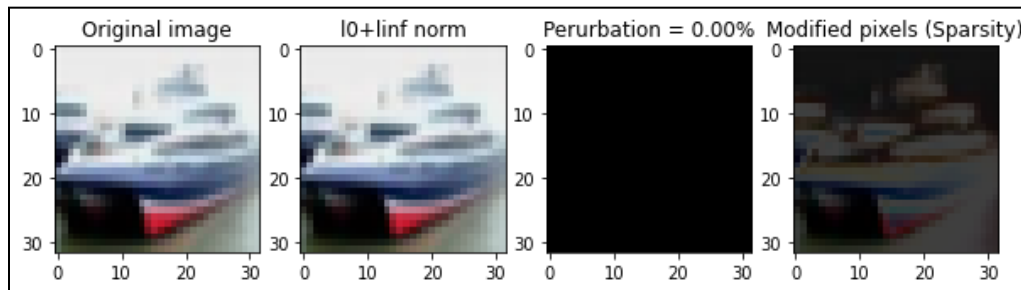
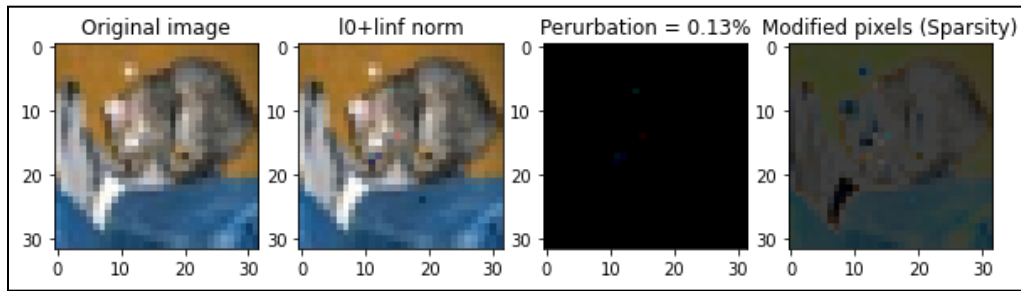
Restart 1 - Robust accuracy: 0.66
Restart 2 - Robust accuracy: 0.64
Restart 3 - Robust accuracy: 0.64
Restart 4 - Robust accuracy: 0.62
Restart 5 - Robust accuracy: 0.6
Pixels changed: [10 0 10 10 0 0 0 10 10 0 0 0 10 0 0 10
0 0 0 0 10 0 10 0
10 0 10 0 0 0 10 10 10 10 0 10 0 10 0 0 0 10 0 0
0 10 10
0 10 0 0 10 10 10 0 0 10 10 10 0 10 0 10 0 0 10
0 10 0
0 0 0 0 0 0 10 0 0 0 0 0 10 10 0 10 0 0 10 0
0 10 10
10 10 0 0]
Robust accuracy at 10 pixels: 60.00%
Maximum perturbation size: 0.10000

```

- Performed PGD attack with L0+Linf norm with changing pixels from 1-50 with n-restarts = 1 and got the following robust accuracy vs number of pixels changed plot.



- Visualized images with PGD attack with L0+Linf norm for sparsity set to 10 pixels.



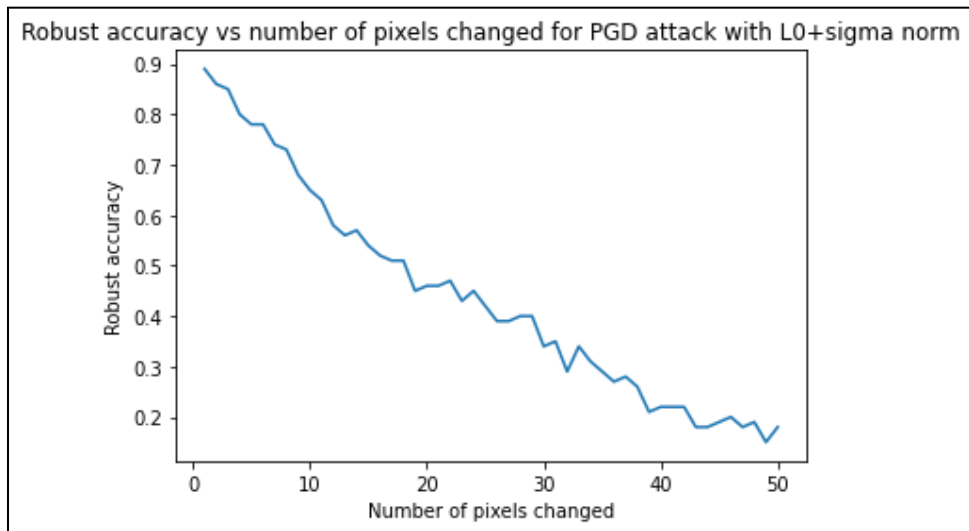
- Performed PGD attack with L0+sigma norm with sparsity = 5 pixels for 5 restarts.

```
Restart 1 - Robust accuracy: 0.81
Restart 2 - Robust accuracy: 0.8
Restart 3 - Robust accuracy: 0.78
Restart 4 - Robust accuracy: 0.77
Restart 5 - Robust accuracy: 0.77
Pixels changed: [0 0 5 0 0 0 0 0 0 0 0 0 0 0 5 0 0 0 0 0 5 0
0 0 5 0 0 0 0 0 5 0 0 5 0
5 0 0 0 0 5 0 0 0 5 5 0 0 0 0 5 5 0 0 0 5 5 5 0 5 0 0 0 0 0 0
0 5 0 0 0
0 0 0 0 5 0 0 0 0 0 5 5 5 5 0 0 0 0 0 0 0 5 0 0 0 0]
Robust accuracy at 5 pixels: 77.00%
Maximum perturbation size: 0.30313
```

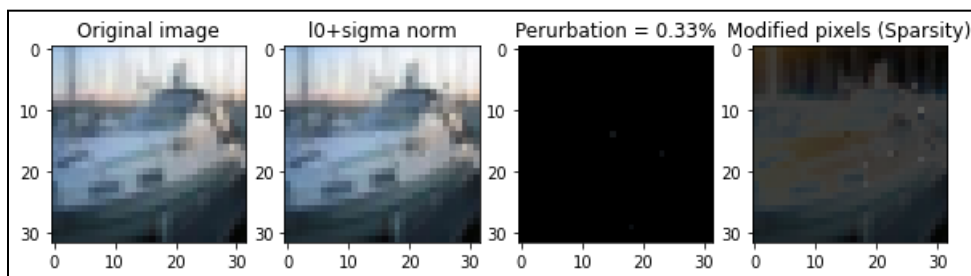
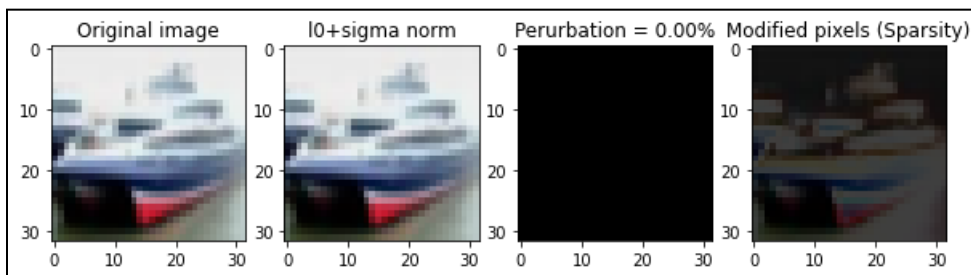
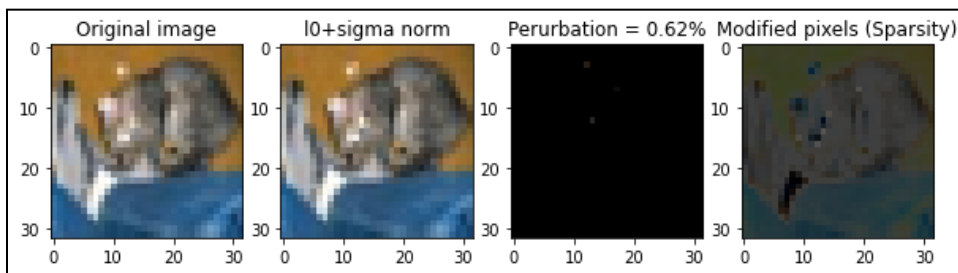
- Performed PGD attack with L0+sigma norm with sparsity = 10 pixels for 5 restarts.

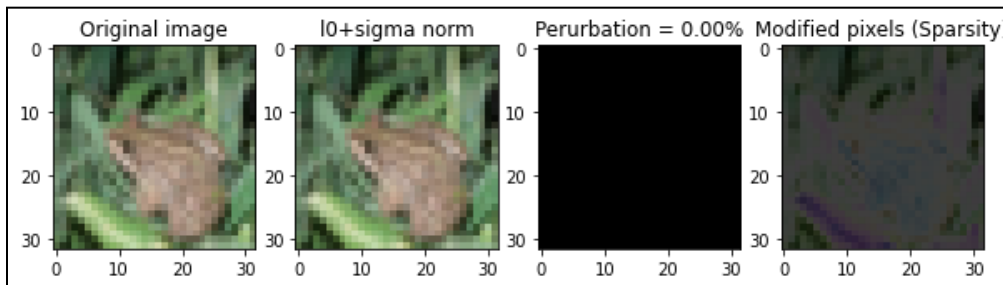
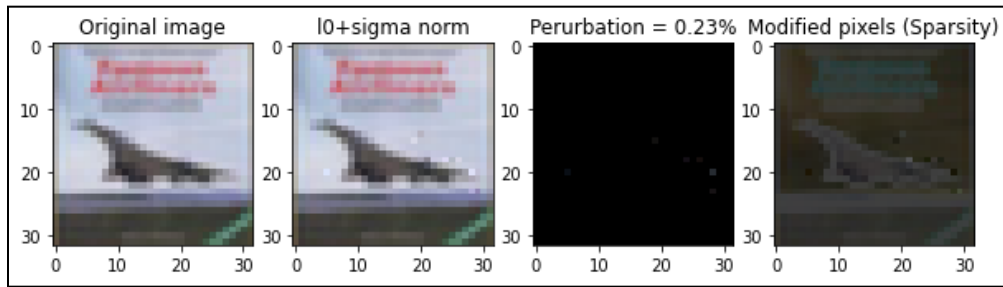
```
Restart 1 - Robust accuracy: 0.61
Restart 2 - Robust accuracy: 0.61
Restart 3 - Robust accuracy: 0.6
Restart 4 - Robust accuracy: 0.6
Restart 5 - Robust accuracy: 0.6
Pixels changed: [10 0 10 10 0 0 0 10 0 10 0 0 10 0 0 10
10 0 0 0 10 0 10 0
10 0 10 0 0 0 0 0 10 0 0 10 0 10 0 0 0 0 10 0 0
0 10 10
0 10 0 0 10 10 0 0 0 10 10 10 0 10 0 10 0 0 0 10
10 10 10
0 0 0 0 0 0 10 0 0 0 0 10 10 10 10 10 0 0 0 10 0
0 10 10
10 0 0 0]
Robust accuracy at 10 pixels: 60.00%
Maximum perturbation size: 0.32860
```

- Performed PGD attack with L0+sigma norm with changing pixels from 1-50 with n-restarts = 1 and got the following robust accuracy vs number of pixels changed plot.



- Visualized images with PGD attack with L0+sigma norm for sparsity set to 10 pixels.

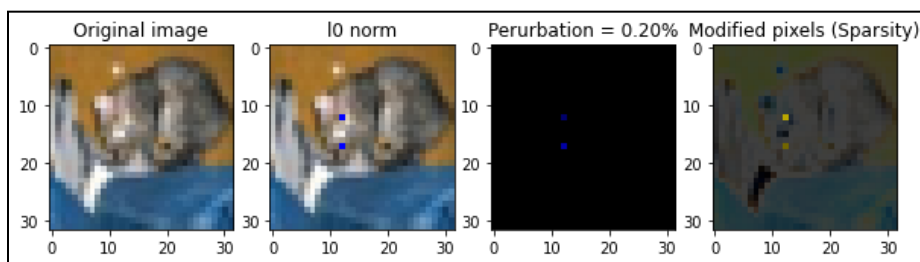




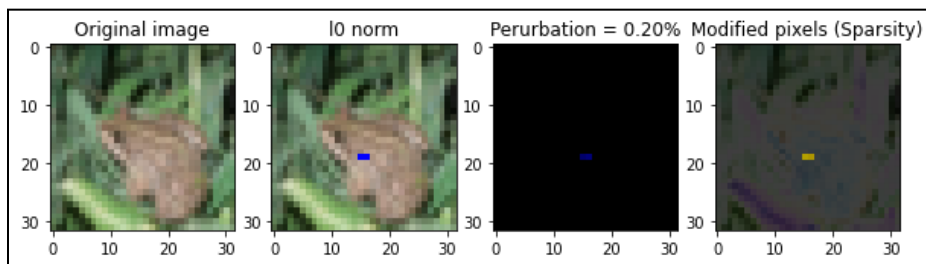
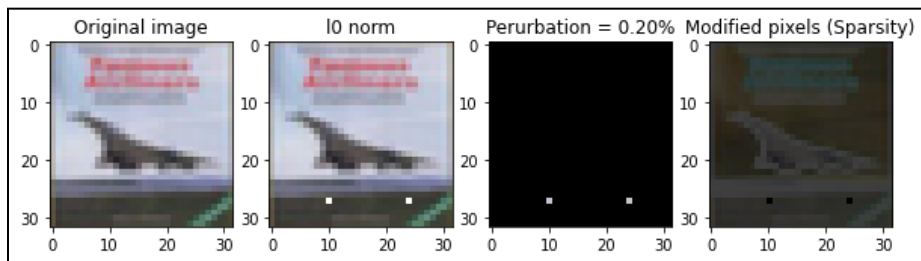
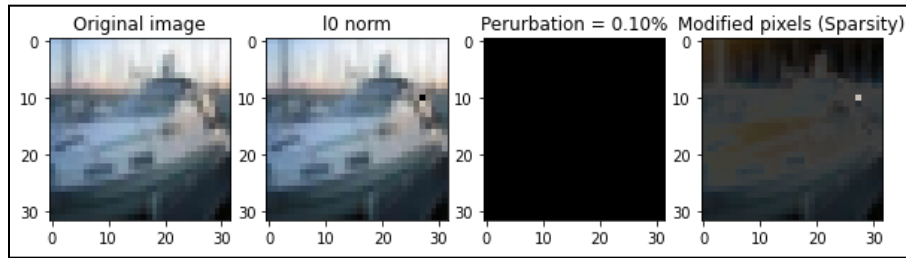
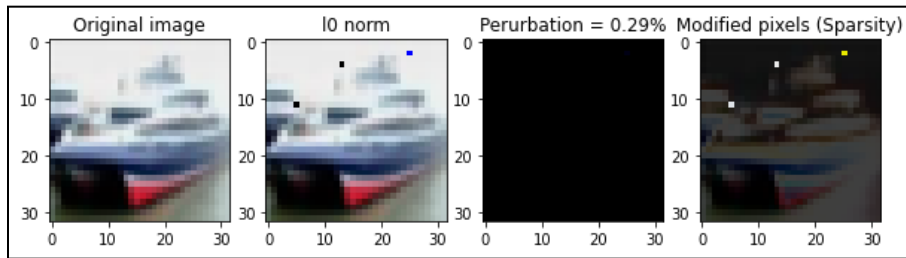
- Then, we moved on to performing CornerSearch attacks.
- CornerSearch attack with L0 norm with  $k = 10$  pixels for  $n\_iter=1000$  and  $N=100$ .

```
Pixels changed: [2 3 1 2 2 2 2 1 1 2 2 0 2 3 5 1 2 2 2 3 1 2 1 0 2 2 1 3 3 3 1 2 1 2 3 1 1
0 0 3 2 2 1 2 3 7 1 0 0 1 4 3 0 1 1 2 4 1 0 0 4 1 1 2 2 5 2 3 2 2 1 2 2 2
2 6 2 1 1 2 2 2 4 2 1 1 0 3 3 4 2 2 2 2 1 1 2 6 2]
Robust accuracy at 10 pixels: 4.00%
Maximum perturbation size: 1.00000
```

- Visualized the images:



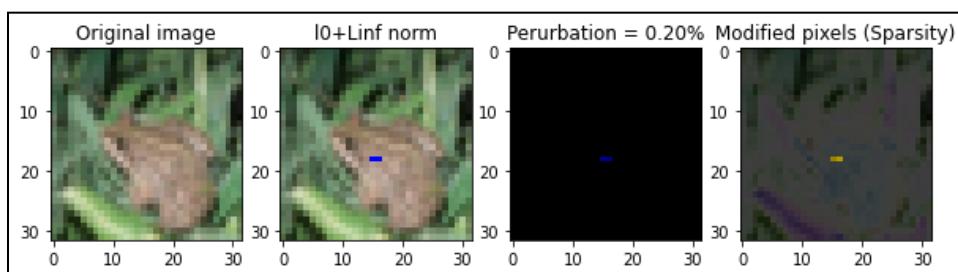
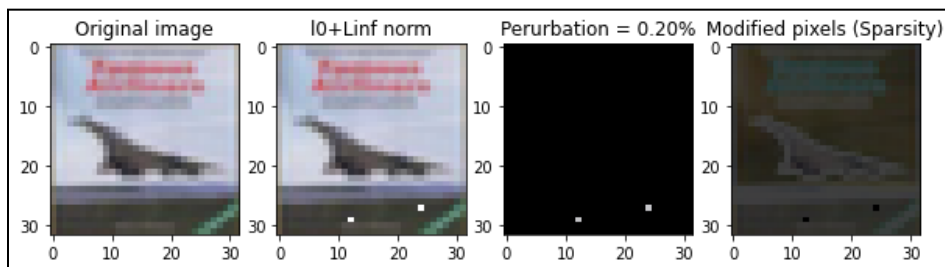
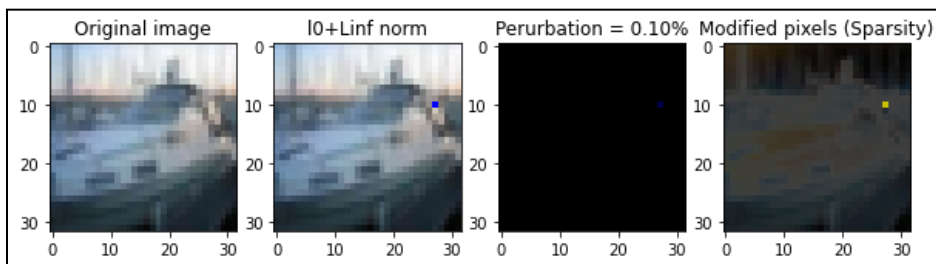
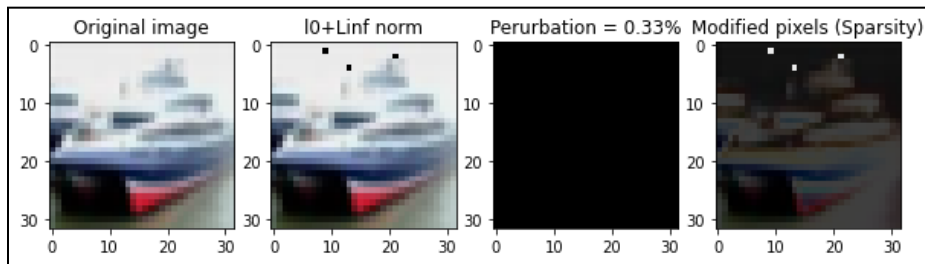
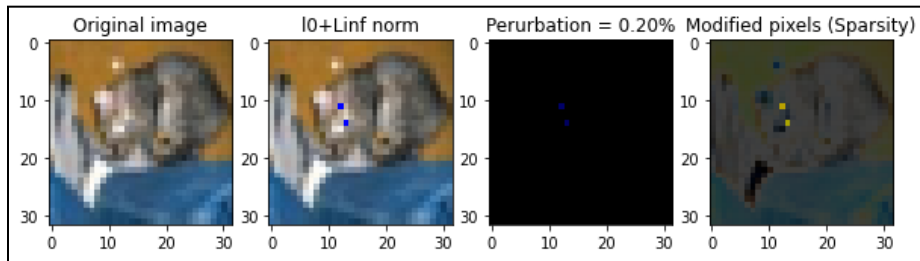




- CornerSearch attack with L0+Linf norm with  $k = 10$  pixels for  $n\_iter=1000$  and  $N=100$ .

```
Pixels changed: [2 3 1 2 2 2 2 1 1 2 2 0 2 3 5 1 2 2 2 3 1 2 1 0 2 2 1 3 3 3 1 2 1 2 3 1 1
0 0 3 2 2 1 2 3 8 1 0 8 1 4 3 0 1 1 2 4 1 0 0 4 1 1 2 2 4 2 3 2 2 1 2 2 2
2 5 2 1 1 2 2 2 5 2 1 1 1 0 3 3 5 2 2 2 2 1 1 2 6 2]
Robust accuracy at 10 pixels: 3.00%
Maximum perturbation size: 1.00000
```

- Visualized the images:



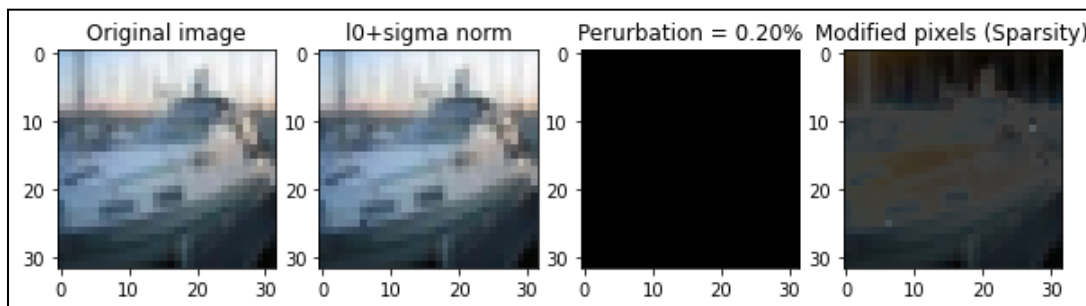
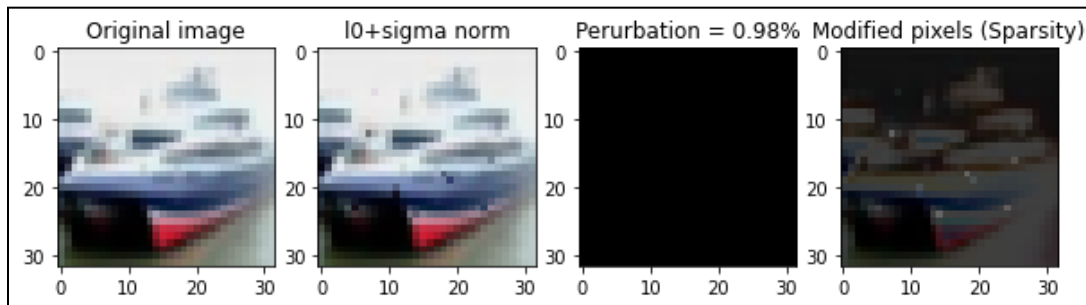
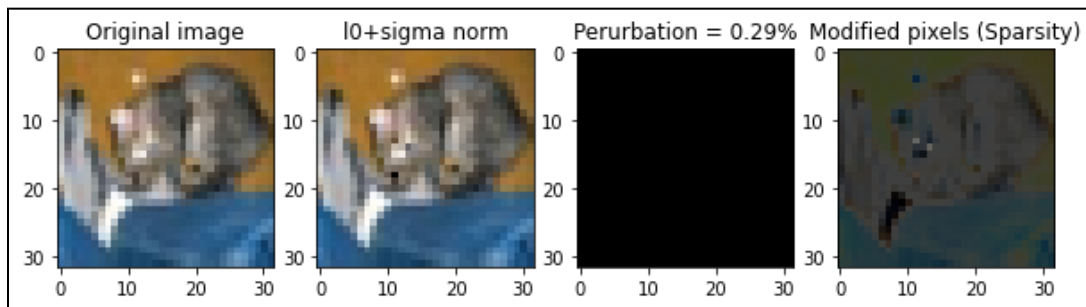
- CornerSearch attack with L0+sigma norm with  $k = 10$  pixels for  $n\_iter=1000$  and  $N=100$ .

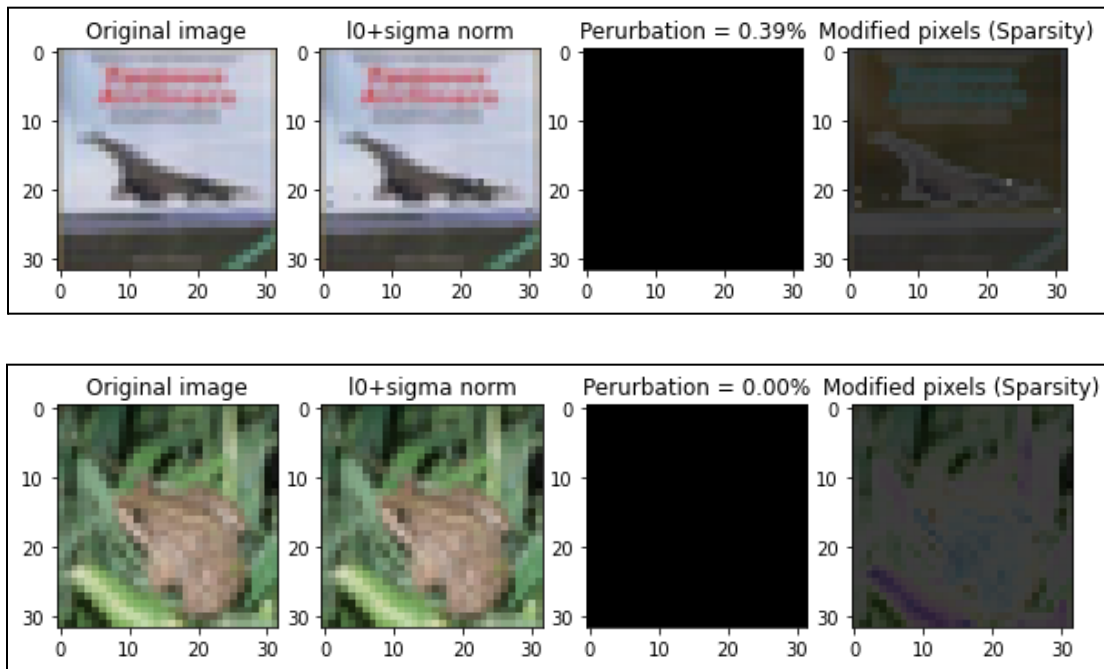
```

Pixels changed: [ 3 10 2 6 0 10 6 2 2 5 6 0 3 0 0 1 6 4 0 0 2 0 1 0
3 8 2 0 7 0 5 4 3 4 0 1 6 0 0 0 10 0 2 0 0 0 2 0
0 3 0 0 0 1 3 8 0 1 0 0 0 1 0 3 0 0 0 0 3 7 2 5
4 9 9 0 5 0 1 10 9 7 0 5 1 2 3 0 0 0 0 3 0 0 3 2
3 0 0 0]
Robust accuracy at 10 pixels: 38.00%
Maximum perturbation size: 0.49077


```

- Visualization of images:





- Therefore, these all were the results that I obtained after applying L0, L0\_Linf, L0+sigma map norms as defined and shown in the Paper to the CornerSearch and PGD attacking methods.
- Then, I tried to get the results on a new dataset FashionMNIST and performed all the necessary data import and data transformation steps.
- Also, built a CNN model for training the FashionMNIST dataset first because there was no pretrained ResNet for FashionMNIST already existing.
- Then, I wrote all the codes and implementation of the same attacks as above like:
  - PGD with L0 norm with changing pixels.
  - PGD with Lo\_Linf norm with changing pixels.
  - PGD with L0+sigma norm with changing pixels.
  - CornerSearch with L0 norm with 10 pixels sparsity.
  - CornerSearch with L0+Linf norm with 10 pixels sparsity.
  - CornerSearch with L0+sigma norm with 10 pixels sparsity.
- Could not run these codes due to running times, GPU constraints the submission deadline of the assignment.

- 
- Therefore, if we run these attacks on FashionMNIST as well, we can again obtain the sparse and imperceivable impacts on the images like we obtained earlier with the CIFAR-10 dataset.