Analyzing and Building Strong Adversarial Attacks

Himanshu Aswani - 160110028
Anubhav Saikia - 180050010
Arjun Kashettiwar - 180050012
Parth Laturia - 180050071
Sunil Kumar Meena - 180050107

Task Description

The following are the different areas of research we focused on while attempting to understand more about adversarial attacks:

1. FGSM (and variants RFGSM, MIFGSM)
2. PGD with Modification 1

We have experimented with certain details that the seminal papers themselves overlooked in our opinion. These revolve around comparing across multiple baseline architectures, changing norms in the objective and different datasets.

Part 1: FGSM

And variants including R-FGSM and MI-FGSM

Attack Set-Up

We have implemented three major variants of FGSM as follows:

1. FGSM (Fast Gradient Sign Method)
   a. This involves taking the sign of the gradient of the loss function with respect to the pixel and adding/subtracting a small epsilon value to that position accordingly.
2. R-FGSM (Random FGSM which involves randomly perturbing the pixels under the standard normal distribution, before performing FGSM)
3. MI-FGSM (Momentum Iterative FGSM which continuously updates the gradients for fixed number of iterations before generating the adversarial image)

Experimental Sections

1. To test our implementations and extrapolate them, we first attacked three popular pre-trained models namely, AlexNet, ResNet 18 and MobileNetV3_Large. We do this as the paper itself focused on only one type of base network. It turns out results are contradictory to expectations.
2. We test the hypothesis of networks being too linear by changing activation functions to sigmoid and tanh.
3. It is known that most networks can be significantly pruned while maintaining performance. We hypothesize that the weights that are eventually pruned may also justify the existence of adversarial examples and test it.
4. One pixel attacks typically involve optimizing an objective function to detect the pixel to attack. We take an opposite stance and explore the importance of each pixel by generating ‘sensitivity maps’ which present interesting visuals.

Results Section 1

Eps = 0	FGSM	RFGSM	MIFGSM
AlxNT	57.98	57.98	57.97
RN18	72.8	72.8	72.8
MNV3	75.73	75.73	75.73

Eps = 0.05	FGSM	RFGSM	MIFGSM
AlxNT	3.79	12.23	
RN18	4.73	10.08	
MNV3	3.91	6.9	

Eps = 0.1	FGSM	RFGSM	MIFGSM
AlxNT	1.04	3.79	
RN18	3.44	4.73	
MNV3	3.79	3.91	

Eps = 0.15	FGSM	RFGSM	MIFGSM
AlxNT	0.54	1.8	
RN18	3.5	3.8	
MNV3	3.62		

Eps = 0.2	FGSM	RFGSM	MIFGSM
AlxNT	0.35	1.04	
RN18	3.47	3.44	
MNV3		3.79	

Eps = 0.25	FGSM	RFGSM	MIFGSM
AlxNT	0.31	0.67	
RN18	3.55	3.34	
MNV3		4.12	

Results Section 2

Eps = 0	FGSM	RFGSM	MIFGSM
ReLU	98	98	98
Sgmd	98.7	98.7	98.7
Tanh	98.03	98.03	98.03

Eps = 0.05	FGSM	RFGSM	MIFGSM
ReLU	96.8	95.52	91.47
Sgmd	89.22	96.05	87.87
Tanh	90.1	95.5	89.28

Eps = 0.1	FGSM	RFGSM	MIFGSM
ReLU	86.7	91.84	70.18
Sgmd	54.29	89.22	45.67
Tanh	69.77	90.16	63.3

Eps = 0.15	FGSM	RFGSM	MIFGSM
ReLU	55.65	84.7	37.02
Sgmd	25.91	74.88	17.57
Tanh	41.31	81.64	30.45

Eps = 0.2	FGSM	RFGSM	MIFGSM
ReLU	23.15	73.55	13.44
Sgmd	13.07	54.29	6.46
Tanh	22.38	69.77	13.85

Eps = 0.25	FGSM	RFGSM	MIFGSM
ReLU	7.73	59.11	3.36
Sgmd	7.07	37.42	2.01
Tanh	12.96	55.41	6.89

Results Section 3

	0	0.05	0.1	0.15	0.2	0.25	0.3
0%	98	96.8	86.7	55.6	23.1	7.73	2.47
10%	98	86.6	55.4	23.2	7.85	2.44	0.9
20%	98	86.5	55.3	23.4	7.61	2.23	0.74
30%	97.9	86.6	55.1	23.4	8.2	2.4	0.83

- The rows represent percentage of pruning in the fully connected layers only - sorted on absolute value and the least p% set to zero. The columns indicate epsilon values.
- Contrary to our expectations, pruning out weights irrelevant to the task does not improve adversarial accuracy - in fact, we should not remove them as they impart adversarial robustness.

Himanshu - Results Section 4

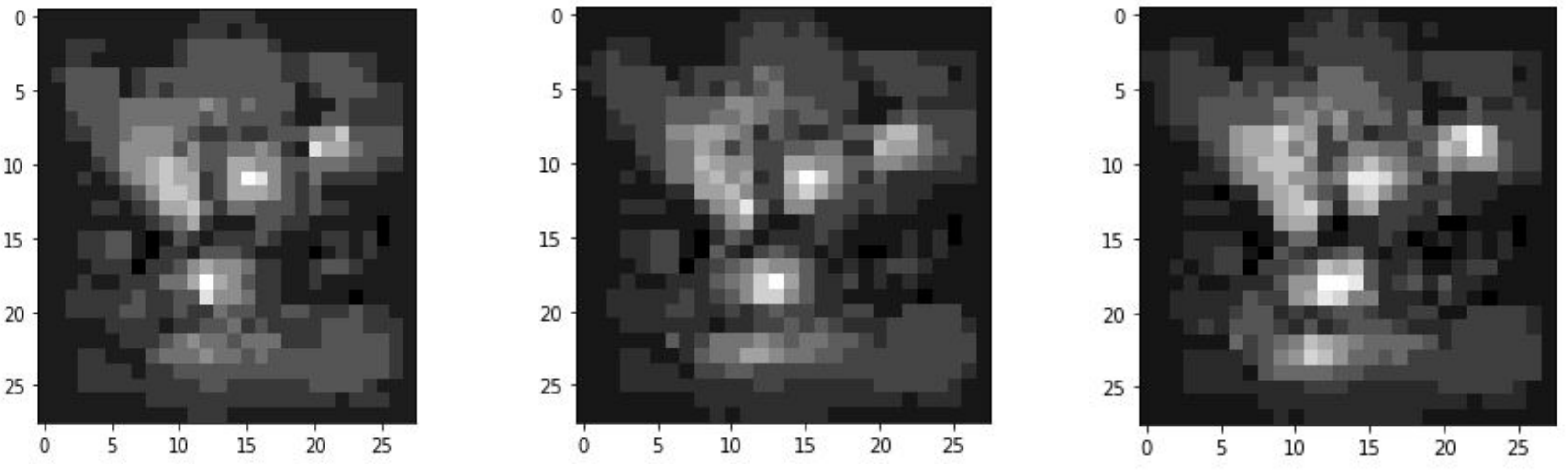


Fig: The sensitivity maps have been generated for class ‘1’ for epsilon values 0.2, 0.25, 0.3. It is interesting to note that irrespective of epsilon values, there emerges a consensus on the spread of pixel importance.

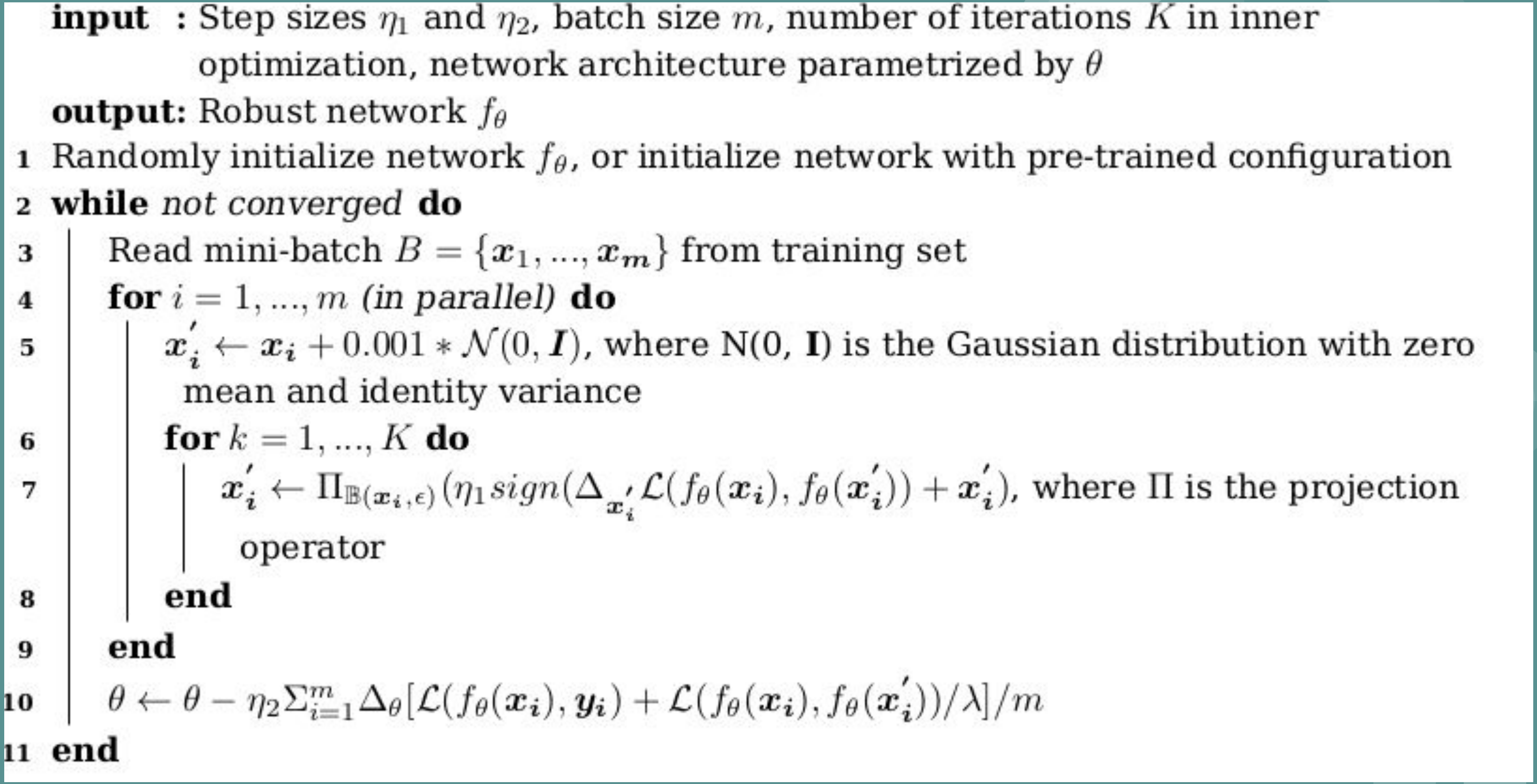
Conclusion

- Results of Section 1 support the hypothesis that model capacity plays a role in robustness - although it appears minor for large epsilon perturbations.
- Results of Section 1 also demonstrate that, for a variety of model types, R-FGSM isn’t as effective as simply FGSM or MI-FGSM(this turns out to be most effective) (The paper performs experiment only on ResNet and InceptionNet architectures.)
- As linearity seems to be a major factor in adversarial examples, we expect non-linear activations to perform comparable if not better than ReLU on adversaries. The results of section 2 indicate the opposite is more likely, Sigmoid provides the least robustness and ReLU the most.
- Generation of sensitivity maps (section 4) may possibly lead to better adversarial example creation as it is unreasonable to assume that a pixel has sufficient influence(white pixels in the map) to determine the predicted label.

Part 2: PGD with Modification 1

Modified Trades Adversarial Training

Original Trades Adversarial Training



```
input  : Step sizes $\eta_1$ and $\eta_2$, batch size $m$, number of iterations $K$ in inner
         optimization, network architecture parametrized by $\theta$
output: Robust network $f_\theta$
1  Randomly initialize network $f_\theta$, or initialize network with pre-trained configuration
2  while not converged do
3      Read mini-batch $B = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_{\boldsymbol{m}}\}$ from training set
4      for $i = 1, ..., m$ (in parallel) do
5          $\boldsymbol{x}_{\boldsymbol{i}}^{'} \leftarrow \boldsymbol{x}_{\boldsymbol{i}} + 0.001 * \mathcal{N}(0, \boldsymbol{I})$, where N(0, I) is the Gaussian distribution with zero
           mean and identity variance
6          for $k = 1, ..., K$ do
7              $\boldsymbol{x}_{\boldsymbol{i}}^{'} \leftarrow \Pi_{\mathbb{B}(\boldsymbol{x}_{\boldsymbol{i}}, \epsilon)}(\eta_1 sign(\Delta_{\boldsymbol{x}_{\boldsymbol{i}}^{'}} \mathcal{L}(f_\theta(\boldsymbol{x}_{\boldsymbol{i}}), f_\theta(\boldsymbol{x}_{\boldsymbol{i}}^{'})) + \boldsymbol{x}_{\boldsymbol{i}}^{'})$, where $\Pi$ is the projection
               operator
8          end
9      end
10     $\theta \leftarrow \theta - \eta_2 \Sigma_{i=1}^{m} \Delta_\theta[\mathcal{L}(f_\theta(\boldsymbol{x}_{\boldsymbol{i}}), \boldsymbol{y}_{\boldsymbol{i}}) + \mathcal{L}(f_\theta(\boldsymbol{x}_{\boldsymbol{i}}), f_\theta(\boldsymbol{x}_{\boldsymbol{i}}^{'}))/\lambda]/m$
11 end
```

TRADES Performance

- ❖ The original TRADES algorithm achieves around 99% natural accuracy as well 99% robust accuracy. Here natural accuracy refers to the accuracy achieved when tested on the test dataset and robust accuracy refers to the one achieved when the test dataset is perturbed as per the FGSM or PGD attack

- ❖ Change of epsilon in the original TRADES algorithm does not affect the robust accuracy of the model. Attacks such as EWR-PGD are able to bring down the robust accuracy to around 92% as reported in the author’s results

Our Modification

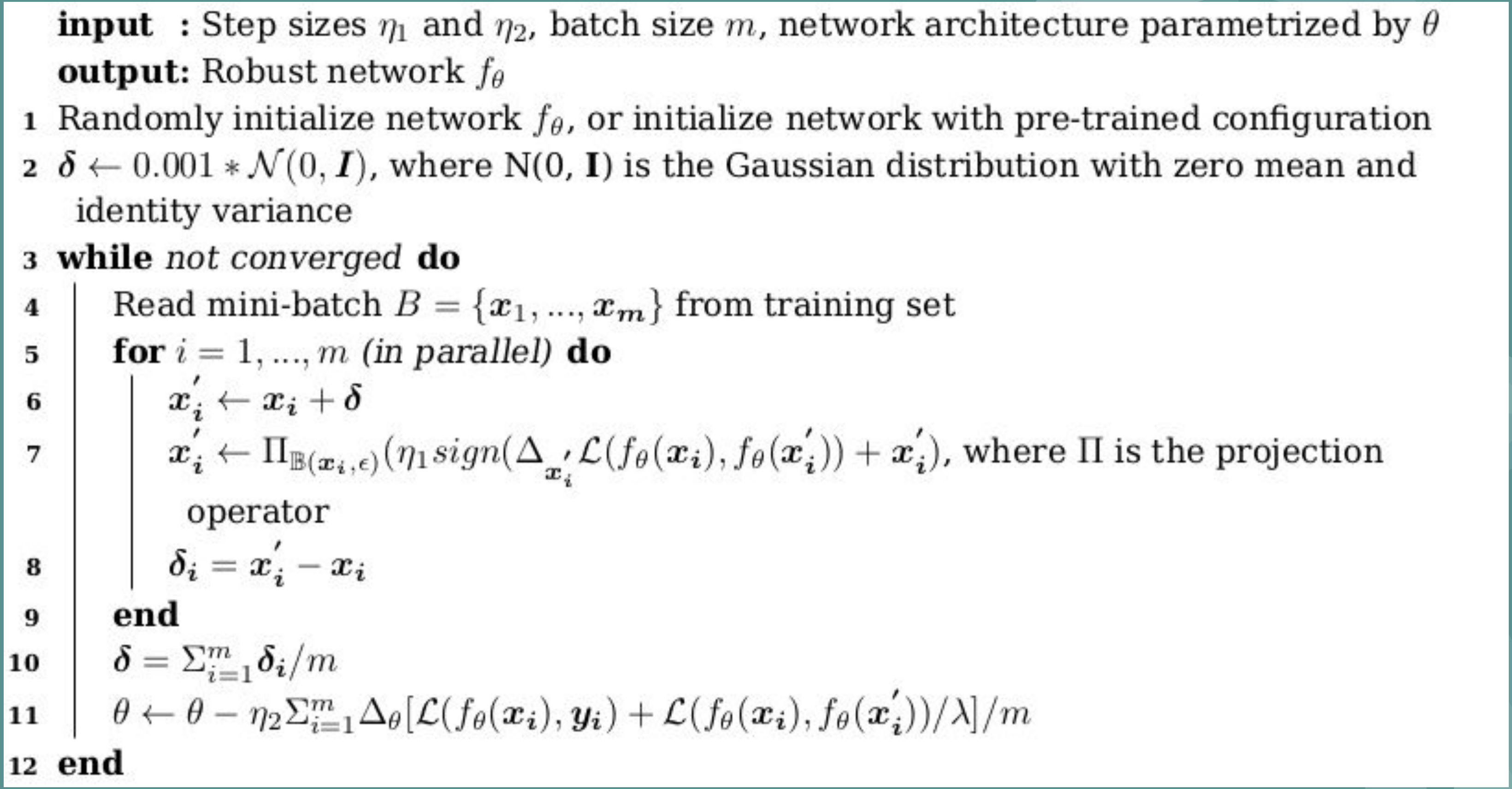
- ❖ The perturbation that was being introduced in the samples originally was from a multivariate gaussian with constant mean and covariance matrix. We decided to instead initialize the perturbation for a mini-batch of samples using the average of the perturbations for each of the samples in the previous mini-batch

- ❖ Along with the above change we also removed the iterative gradient step (an inner loop) and calculated the perturbed samples using only one gradient step

- ❖ The intuition behind this is that non zero perturbations for samples based on the perturbations for previous samples should help find a point of maximum loss during adversarial training. This should depend on how large epsilon is however and that becomes apparent from the graph in the next slide

- ❖ This intuition was inspired by the work done on FAST-FGSM previously. This kind of a modification will help make adversarial training much faster and consume less resources

Modified Trades Method



```
input  : Step sizes $\eta_1$ and $\eta_2$, batch size $m$, network architecture parametrized by $\theta$
output: Robust network $f_\theta$
Randomly initialize network $f_\theta$, or initialize network with pre-trained configuration
$\boldsymbol{\delta} \leftarrow 0.001 * \mathcal{N}(0, \boldsymbol{I})$, where N(0, I) is the Gaussian distribution with zero mean and
 identity variance
while not converged do
    Read mini-batch $B = \{\boldsymbol{x}_1, ..., \boldsymbol{x_m}\}$ from training set
    for $i = 1, ..., m$ (in parallel) do
        $\boldsymbol{x}_i^{'} \leftarrow \boldsymbol{x_i} + \boldsymbol{\delta}$
        $\boldsymbol{x}_i^{'} \leftarrow \Pi_{\mathbb{B}(\boldsymbol{x_i}, \epsilon)}(\eta_1 sign(\Delta_{\boldsymbol{x}_i^{'}} \mathcal{L}(f_\theta(\boldsymbol{x_i}), f_\theta(\boldsymbol{x}_i^{'})) + \boldsymbol{x}_i^{'})$, where $\Pi$ is the projection
         operator
        $\boldsymbol{\delta_i} = \boldsymbol{x}_i^{'} - \boldsymbol{x_i}$
    end
    $\boldsymbol{\delta} = \Sigma_{i=1}^{m} \boldsymbol{\delta_i}/m$
    $\theta \leftarrow \theta - \eta_2 \Sigma_{i=1}^{m} \Delta_\theta[\mathcal{L}(f_\theta(\boldsymbol{x_i}), \boldsymbol{y_i}) + \mathcal{L}(f_\theta(\boldsymbol{x_i}), f_\theta(\boldsymbol{x}_i^{'}))/\lambda]/m$
end
```

CIFAR 10 Experiment Details and Main Results

Training HyperParameters:

batch_size: 128, weight_decay: 2e-4, learning_rate: 0.1, SGD Momentum: 0.9, random seed: 1, epochs: 7

Inner Maximization Perturbation:

l∞-norm,epsilon: 0.031, num_steps: 1, step_size: 0.007,beta: 6.0 #regularization (1/lambda) in TRADES

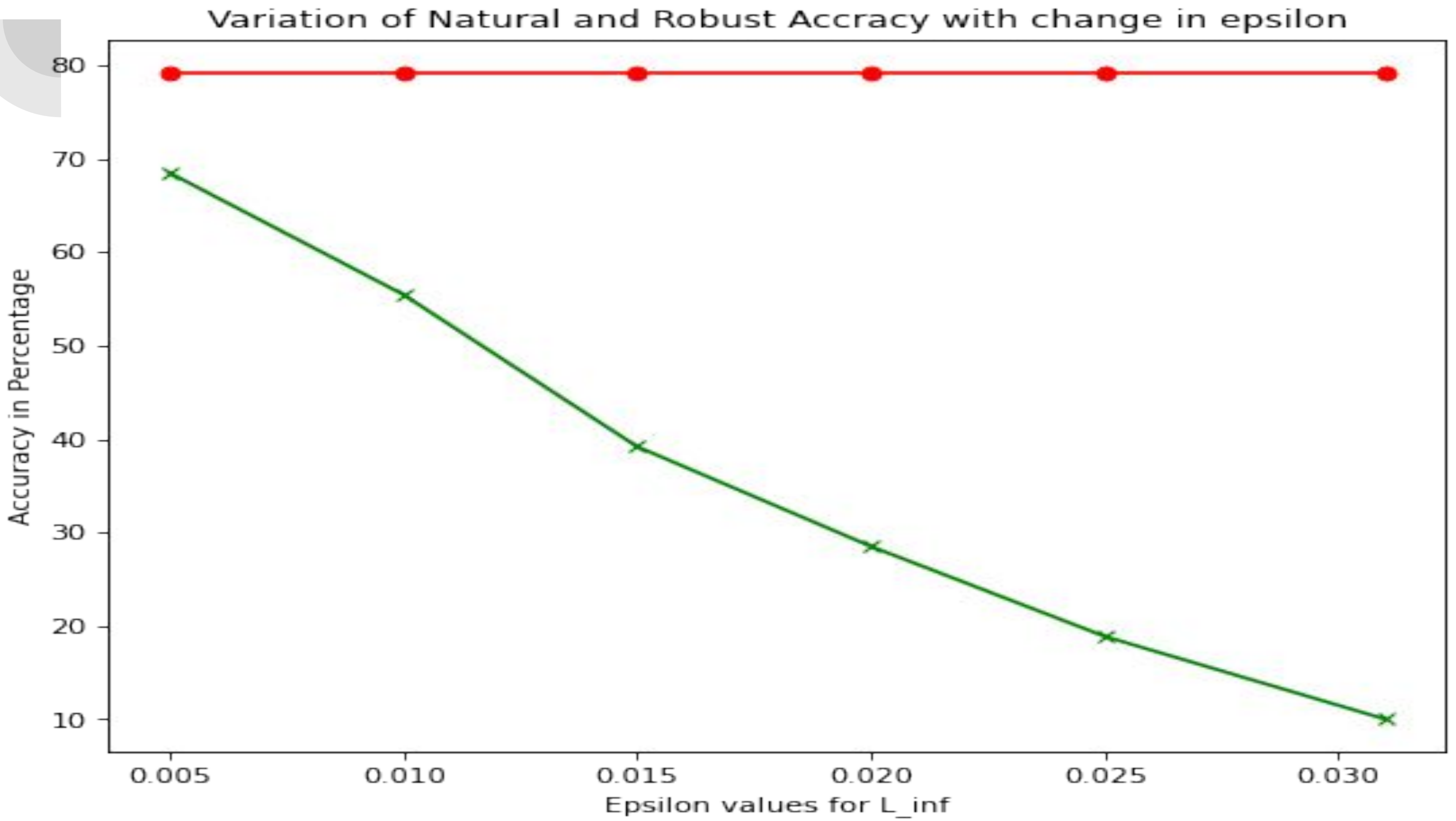
Model: WideResNet(Describe?), Total Training Time: 3 hours 30 minutes

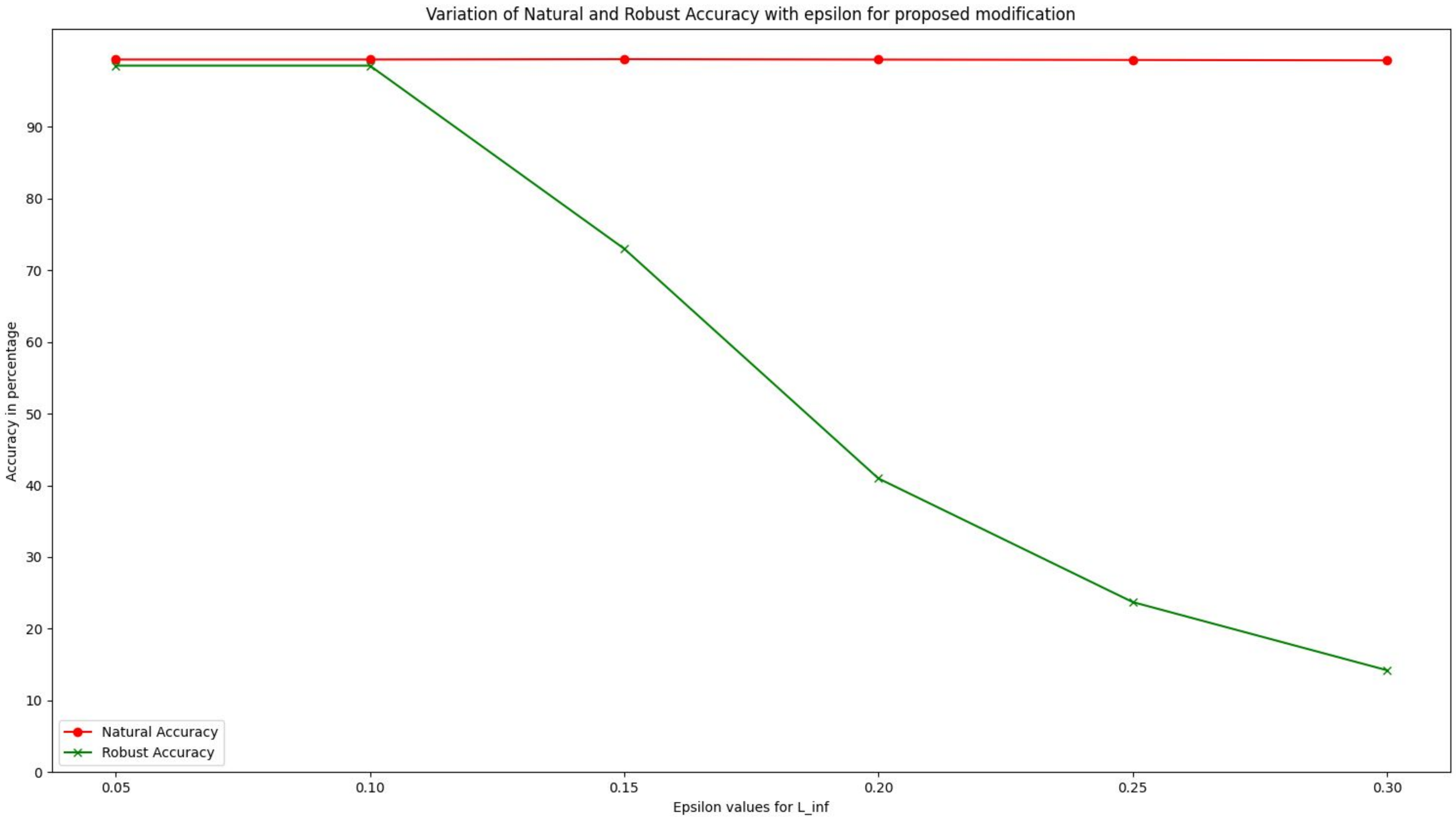
Result of different Attacks on The Model:

Attack	WideResnet Model trained with original trades (76 epochs)		WideResnet Model trained with modified trades (7 epochs)	
	Natural Accuracy	Robust Accuracy	Natural Accuracy	Robust Accuracy
FGSM	84.92%	61.06%	79.16%	64.58%
FGSM-20	84.92%	56.61%	79.16%	10.02%

Robust Accuracy => Accuracy achieved with data perturbed by attack

FGSM-20 attack with different epsilons




Variation of Natural and Robust Accuracy with epsilon for proposed modification
Accuracy in percentage
0
10
20
30
40
50
60
70
80
90
0.05
0.10
0.15
0.20
0.25
0.30
Epsilon values for L_inf
Natural Accuracy
Robust Accuracy

Observations and Conclusions

- The modification performs quite well for small values of epsilon (less than or equal to 0.1). Robust accuracy values are at par with natural accuracy values which is also the case for standard adversarial training methods. However for larger values of epsilon the robust accuracy decreases sharply and adversarial performance deteriorates

- Larger epsilon values naturally require a larger search space to be explored which is why the non zero initialization based on perturbations of previous mini batches and single gradient step fail for these cases

- Instead of using same delta for whole batch if we have used a multivariate gaussian with mean of delta, we may have gotten better results

Related Work

- https://github.com/fra31/auto-attack
- https://github.com/Harry24k/adversarial-attacks-pytorch
- https://pytorch.org/tutorials/beginner/fgsm_tutorial.html
- Explaining and Harnessing Adversarial Examples https://arxiv.org/pdf/1412.6572.pdf
- One pixel attacks for fooling deep neural networks https://arxiv.org/pdf/1710.08864.pdf
- RFGSM attack https://arxiv.org/pdf/1705.07204.pdf
- Boosting Adversarial Attacks with Momentum https://arxiv.org/pdf/1710.06081.pdf
- Trades - https://arxiv.org/pdf/1901.08573.pdf
- Fast vs Free Adversarial Training - https://arxiv.org/pdf/2001.03994.pdf
- Auto PGD Attack - https://arxiv.org/abs/2003.01690
- Some improvements have been suggested for methods such as the one presented above. In particular, addition of a regularizer called *Grad_align* helps make the decrease in robust accuracy slower but accuracy remains significantly lower than standard training methods.

References to existing code and libraries for the project

Our Github Repository - https://github.com/ParthLa/CS726_Project

THANK YOU !!!!!!!!