Adversarial Examples for CNNs

Julian D'Costa & Gaurang Sriramanan
Indian Institute of Science

Abstract

Despite their superhuman ability to recognize images, CNNs are still easily fooled by adversarial examples, a problem of growing importance as image recognition expands into areas where reliability and security are paramount. We study robust optimization, a training method based on minimizing worst-case loss for a given threat model. We also consider the Wasserstein metric as an alternative threat model to the standard L_p balls, as a way to capture a different aspect of "intuitively small" perturbations, examining the theory behind computing fast approximations to the notoriously intractable Wasserstein metric. Our experimental work demonstrates the transferability and stability of perturbations across images and networks and the robustness of adversarial training across threat models, in particular displaying the extensive robustness of Wasserstein-trained networks.

1 Introduction

Adversarial examples for CNNs were studied in the early papers *Intriguing properties of neural networks* [Szegedy et al. (2013)] and *Explaining and Harnessing Adversarial Examples* [Goodfellow et al. (2014)]. They showed that it was easy to generate an image that was only imperceptibly different from an image that the neural network classified correctly, yet this perturbed image would be wrongly classified. These papers used a constraint that the perturbation must be small in the l_∞ norm, which meant that no pixel could be changed by more than a given small epsilon.

An easy way to generate these examples is by Fast Gradient Sign Method (FGSM), which was one of the first methods used to generate Adversarial examples in the DL community [Goodfellow et al. (2014)].It tries to move the image in the direction of the gradient of the loss function, which maximises the loss, subject to the constraint that no pixel can change more than epsilon. This leads to setting the perturbation

$$\boldsymbol{\eta} = \epsilon \operatorname{sign}\left(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y)\right),$$

assuming the function is linear enough (or that epsilon is small enough) that the corner of the l_∞-norm ball is the closest point in the constraint set to the loss maximising perturbation. A short analysis shows that FGSM is the *exact optimal attack* against a linear, binary classification model under l_∞ constraints. Thus, FGSM assumes a linear approximation of the neural network for a pertubation $\delta \in (-\epsilon, \epsilon)$. The following image is an example of FGSM we implemented:

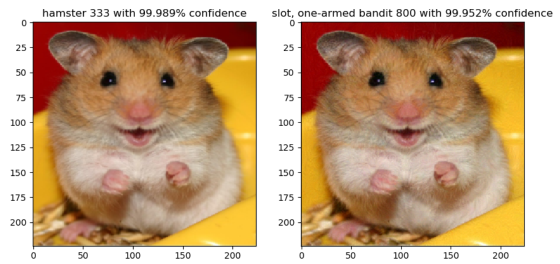

hamster 333 with 99.989% confidence
slot, one-armed bandit 800 with 99.952% confidence
0
25
50
75
100
125
150
175
200
0
50
100
150
200

Figure 1: The FGSM attack was run using $\epsilon = 0.005$, using the ResNet18 convolutional network pretrained on ImageNet. We used the Default Pretrained ResNet18 available on PyTorch

2 Robust Optimization

While it is possible to defend networks from FGSM based attacks by training on them (adversarial training, i.e. adding adversarial images to the training set), these networks are still vulnerable to other attacks. A more principled approach to training robust networks was introduced in *Towards deep learning models resistant to adversarial attacks* [Madry et al. (2017)].

They suggest that instead of adding more and more adversarial images to the training set in reaction to each new attack, one should start with a concrete threat model to defend against, and optimize a guarantee that the network will remain robust against any attack that falls within the parameters of the threat model.

The new loss function is defined as follows: We now allow an adversary to perturb the sample $\mathbf{x}$ before feeding it to our classifier f. Since we are primarily interested in perturbations that do not change the true label, we limit the adversary by restricting it to a pre-defined perturbation set $P(x)$. We want to minimize the worst possible performance in this set. This leads to the following quantity:

$$\mathbb{E}_{x,y\sim D}\left[\max_{x'\in P(x)} L\left(f\left(x'\right),y\right)\right].$$

Note that since this robust loss does not specify a specific attack method, if we get a low robust loss, we can be confident that no attack at the given data point with the threat model $P(x)$ will succeed. The question is, what is a good perturbation set $P(x)$, and how can we optimize the robust loss?

Madry et al take the perturbation set $P(x)$ to be the well-studied l_∞ epsilon ball, which generates imperceptible perturbations as long as epsilon is small.

2.1 Training the Robust Model

The empirical version of the Robust loss

$$\mathbb{E}_{x,y\sim D}\left[\max_{x'\in P(x)} L\left(f\left(x'\right),y\right)\right],$$

is

$$\min_\theta \frac{1}{n}\sum_{i=1}^{n}\max_{x'\in P(x_i)} L\left(f_\theta\left(x'\right),y_i\right).$$

In order to optimise this function with SGD, we need to compute gradients of the robust loss

$$\phi_{x,y}(\theta)=\max_{x'\in P(x)} L\left(f_\theta\left(x'\right),y\right).$$

Note that ϕ is itself an optimization problem, so we can't simply run backpropagation. Fortunately, we can apply Danskin's theorem, which tells us that

$$\nabla_\theta\phi_{x,y}(\theta)=\nabla_\theta L\left(f_\theta\left(x^*\right),y\right),$$

if x^* is a maximiser of the inner max problem.

Thus getting the gradient reduces to finding the maximiser, which is the "worst" adversarial point. Madry et al use Projected Gradient Descent to find these points. By running PGD on 10^5 random seeds, and getting relatively similar loss values, they empirically checked that there are no points with much higher loss to be found, so PGD is finding points close to the true maximisers. While Danskin's theorem is only valid at the true maximiser, this result gives us confidence that the gradient will give us relatively good updates.

Projected Gradient Descent is an iterative algorithm for constrained optimization that involves taking a gradient step and projecting back into the constraint set, for example the l_∞ update: $x'=\Pi_{P(x)}(x-\eta\cdot\mathrm{sign}(\nabla L(x,y)))$, and repeating this procedure until convergence.

This gives the following algorithm for robust training:

1. Sample a data point x, y.
2. Compute the maximizer x of the robust loss $\phi_{x,y}(\theta)$

3. Compute the gradient $g = \nabla_\theta L\left(f_\theta\left(x^*\right), y\right)$
4. Update θ with the gradient g
5. Repeat steps 1-4 until convergence.

Practically, we find some convergence issues while utilising just the SGD algorithm for implementing PGD attacks. Madry et al argue that this is due to the small initial norm of the gradients, which poses a problem as increasing the learning rate to counteract the small initial gradients causes clipping just like in FGSM in later iterations. To solve the problem, they suggest using normalised steepest descent method, which has the update

$$z := z - \underset{\|v\| \leq \alpha}{\text{argmax}} v^T \nabla_z f(z).$$

We attempted an alternative fix: we decided to utilise the Adaptive Moment Estimation method (ADAM) for implementing PGD, as it uses running averages of both the gradients, as well as the second moments of the gradients which helps alleviate the problem of small initial gradients, by using the normalising and forgetting factors:

$$m_w^{(t+1)} \leftarrow \beta_1 m_w^{(t)} + (1-\beta_1)\nabla_w L^{(t)}$$

,

$$v_w^{(t+1)} \leftarrow \beta_2 v_w^{(t)} + (1-\beta_2)(\nabla_w L^{(t)})^2,$$

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \frac{\hat{m}_w}{\sqrt{\hat{v}_w} + \epsilon}.$$

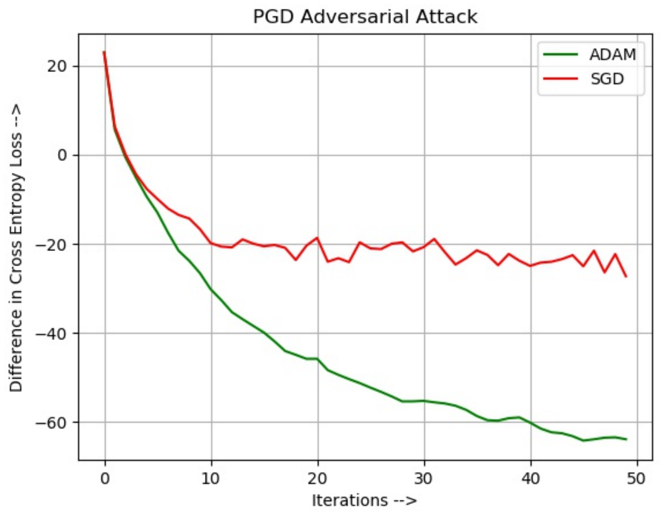


Figure 2: Our convergence results for PGD with SGD and ADAM

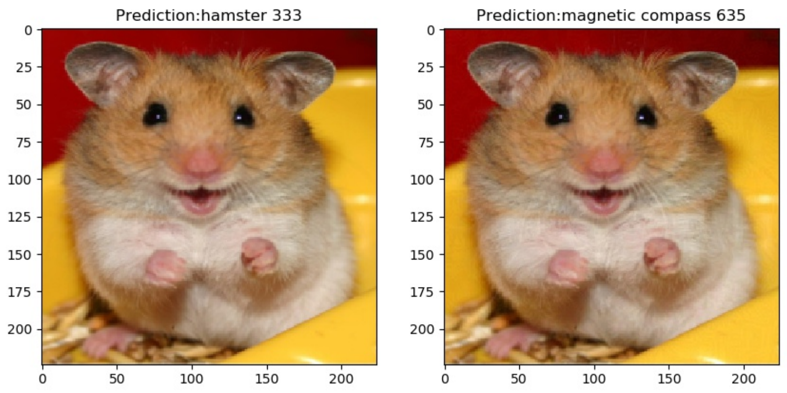
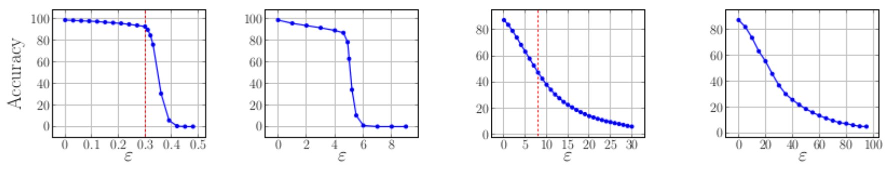


Figure 3: We ran a PGD attack with 50 iterations and epsilon = 2/255 on a ResNet18 convolutional network pretrained on ImageNet

2.2 Results of PGD adversarial training



(a) MNIST, ℓ_∞ norm (b) MNIST, ℓ_2 norm (c) CIFAR10, ℓ_∞ norm (d) CIFAR10, ℓ_2 norm

Figure 6: Performance of our adversarially trained networks against PGD adversaries of different strength. The MNIST and CIFAR10 networks were trained against $\varepsilon = 0.3$ and $\varepsilon = 8$ PGD ℓ_∞ adversaries respectively (the training ε is denoted with a red dashed lines in the ℓ_∞ plots). We notice that for ε less or equal to the value used during training, the performance is equal or better. For MNIST there is a sharp drop shortly after.

Figure 4: Madry et al's results show that an l_∞ trained network achieves 89% accuracy on MNIST and 45% accuracy on CIFAR10 against a PGD based l_∞ attack

Madry et al ran a challenge inviting people to break their PGD trained networks, but no attack was able to reduce accuracy on CIFAR10 to less than 41%, supporting their claim that PGD is the 'ultimate first-order adversary'.

3 Wasserstein Adversarial Examples

3.1 The Wasserstein metric

Individual pixels cannot change very much in l_p norm balls, but there are transformations, like translations and rotations that look small to humans since they preserve the large scale structure of the image, even though individual pixels may change a lot. A metric that captures some of this intuition is the Wasserstein metric, which is defined as a distance between two probability distributions μ and ν by

$$W_p(\mu,\nu) := \left(\inf_{\gamma\in\Gamma(\mu,\nu)} \int_{M\times M} d(x,y)^p \, \mathrm{d}\gamma(x,y) \right)^{1/p}$$

This quantity can be intuitively understood as the minimum cost of moving probability mass to change one distribution into another [see Villani (2008) for details]. When applied to images, this can be interpreted as the cost of moving pixel mass from one pixel to another, where the cost increases with distance. Let x and y be two images, considered as points in $\mathbb{R}^n_+$ with coefficients summing to 1, and let $C \in \mathbb{R}^n$ be some non-negative cost matrix where C_{ij} encodes the cost of moving mass from x_i to y_j . Then, the Wasserstein distance d_W between x and y is defined to be

$$d_{\mathcal{W}}(x,y) = \min_{\Pi\in\mathbb{R}^{n\times n}_+} \langle \Pi, C \rangle$$

$$\text{subject to } \Pi 1 = x, \Pi^T 1 = y,$$

where the minimization is over transport plans Π, whose entries Π_{ij} encode how much mass moves from x_i to y_j.

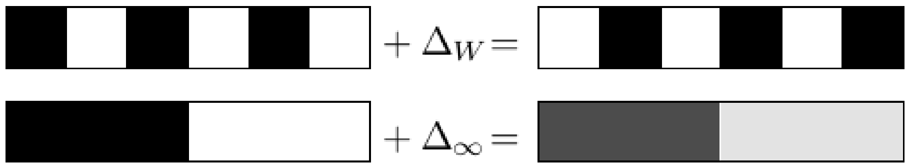


Figure 5: Wasserstein vs l_p

To illustrate the difference between l_p norms and the Wasserstein metric, consider the image above. The top example utilizes a perturbation ΔW to shift the image one pixel to the right, which is small with respect to Wasserstein distance since each pixel moved a minimal amount, but large with respect to l_∞ distance since each pixel changed a maximal amount. In contrast, the bottom example utilizes a perturbation $\Delta\infty$ which changes all pixels to be grayer. This is small with respect to l_∞ distance, since each pixel changes by a small amount, but large with respect to Wasserstein distance, since the mass on each pixel on the left had to move halfway across the image to the right.

3.2 Projection onto the Wasserstein ball

The difficulty with using the Wasserstein ball as our constraint set $P(x)$ is that projecting onto the Wasserstein ball is itself an optimisation problem, unlike the case of l_p, where there exists a closed form solution.
The paper Wasserstein Adversarial Examples via Projected Sinkhorn Iterations [Wong et al. (2019)] uses the technique of adding an entropy regularisation term for the transport plan Π, and subsequently using a modified version of the Sinkhorn-Knopp scaling algorithm (for converting a matrix with positive entries to a doubly-stochastic matrix by pre- and post-multiplying by diagonal matrices) which approximately minimises the Wasserstein metric as was shown earlier in [Cuturi (2013)].
The key proposal in the paper is to solve:

$$\underset{z\in\mathbb{R}^n_+,\Pi\in\mathbb{R}^{n\times n}_+}{\text{minimize}}\ \frac{1}{2}\|w-z\|_2^2+\frac{1}{\lambda}\sum_{ij}\Pi_{ij}\log\left(\Pi_{ij}\right)$$

$$\begin{array}{c}\text{subject to }\Pi 1 = x, \Pi^T 1 = z\\ \langle \Pi, C\rangle \leq \epsilon\end{array}.$$

In order to actually solve the constrained minimisation problem, they solve the Dual Problem, by maximising g, the infimum of the Lagrangian:

$$g(\alpha,\beta,\psi) = -\frac{1}{2\lambda}\|\beta\|_2^2 - \psi\epsilon + \alpha^T x + \beta^T w$$
$$-\sum_{ij}\exp\left(\alpha_i\right)\exp\left(-\psi C_{ij}-1\right)\exp\left(\beta_j\right)$$

Observe, the dual variables α and β correspond to the equality constraints $\Pi 1 = x$ and $\Pi^T 1 = z$, while the positive dual variable ψ corresponds to the inequality constraint $\langle \Pi, C\rangle \leq \epsilon$.
The Karush-Kuhn-Tucker optimality conditions (similarly used in SVM's) are now used to find the analytical form of g in terms of optimal z^* and $(\Pi_{ij})^*$. The optimal values for $(\alpha^*, \beta^*, \psi^*)$ that maximise the dual, are then assured to be the corresponding optimal values for *minimising* the Primal problem.
The optimal values α^* and β^* can be found analytically, with the latter requiring the Lambert-W function. However, the optimal ψ^* cannot be solved for analytically, and so the paper uses Newton iteration till convergence for solving the Dual problem, while simultaneously ensuring the positivity constraint of ψ.

This algorithm produces a point that may not be the optimal projection (since we are minimizing a different objective), but the point is guaranteed to lie within the Wasserstein ball, which means that all the attacks do belong in the constraint set.

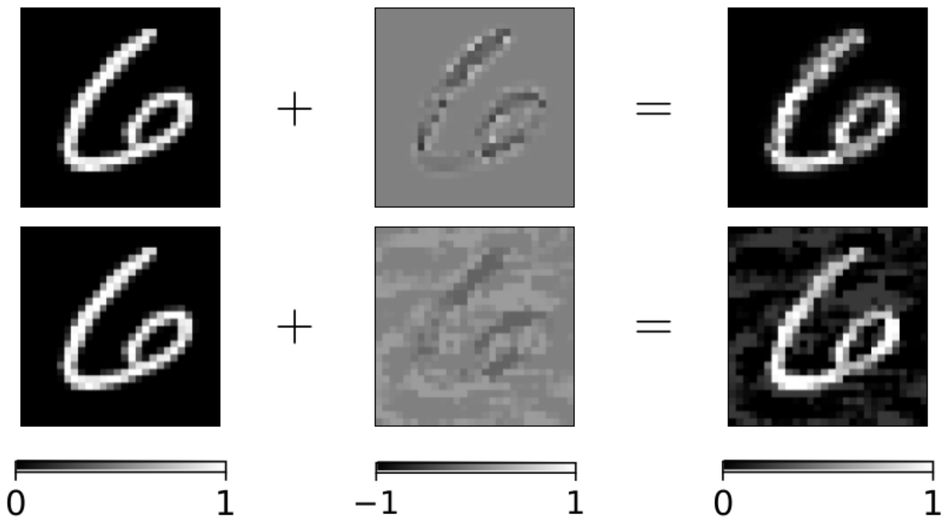
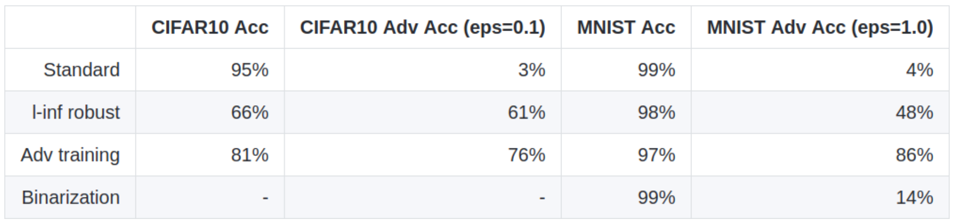


Figure 6: A comparison of a Wasserstein (top) vs an l_∞ (bottom) adversarial example for an MNIST classifier (for $\epsilon = 0.4$ and 0.3 respectively), showing the original image (left), the added perturbation (middle), and the final perturbed image (right). The Wasserstein perturbation has a structure reflecting the actual content of the image, whereas the l_∞ perturbation also attacks the background pixels. From Wong et al. (2019)

3.3 Wasserstein results



	CIFAR10 Acc	CIFAR10 Adv Acc (eps=0.1)	MNIST Acc	MNIST Adv Acc (eps=1.0)
Standard	95%	3%	99%	4%
l-inf robust	66%	61%	98%	48%
Adv training	81%	76%	97%	86%
Binarization	-	-	99%	14%

Figure 7: Wasserstein attacks on various networks

Wong et al attack a standard network, an l_∞ robust network and a network trained with Wasserstein adversarial examples. They find that l_∞ robustness transfers substantially to Wasserstein attacks.

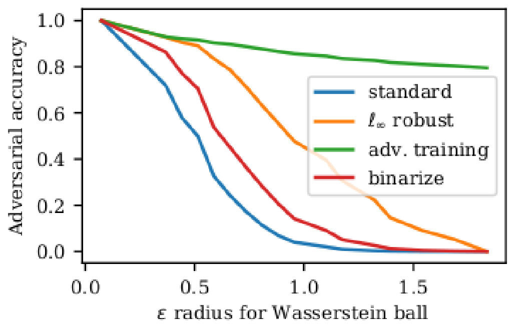


Figure 8: Fooling rates for Wasserstein attacks on MNIST

Notice that Wong et al were not able to attack the adversarially trained network down to 0% accuracy. While this may be because they did not try large enough values of epsilon, it raises the possibility that the Sinkhorn iteration with entropy regularisation is not a good enough approximation to generate strong adversarial examples.

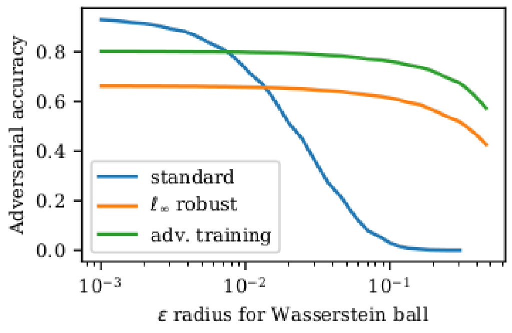
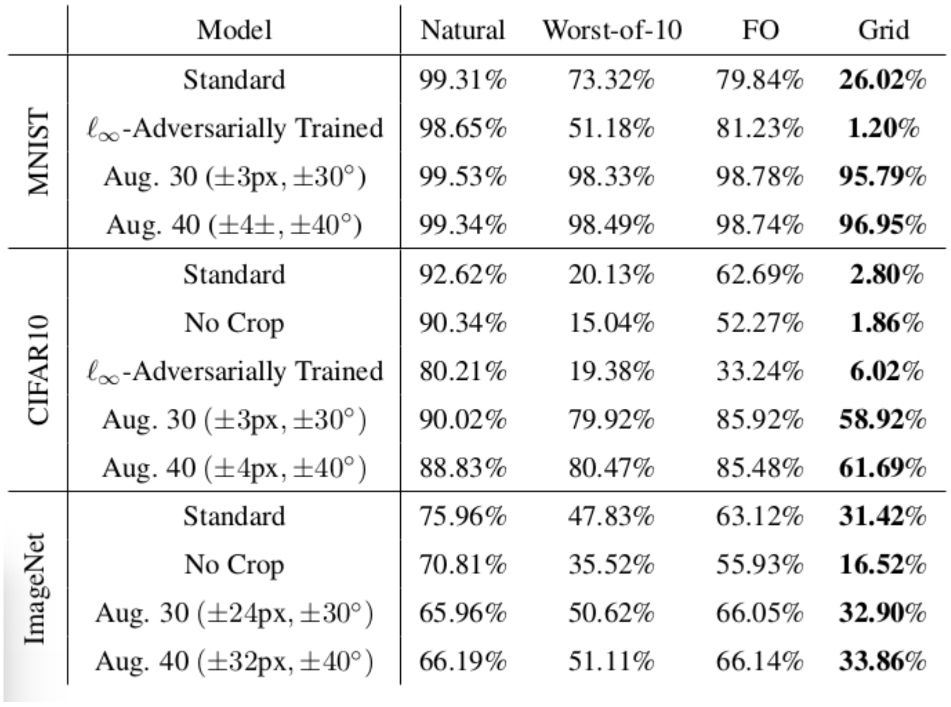


Figure 9: Fooling rates for Wasserstein attacks on CIFAR10

4 Transferability of Robustness

While Wong et al found that l_∞ robustness conferred substantial protection against Wasserstein attacks, we also looked at the paper A Rotation and a Translation Suffice: Fooling CNNs with Simple Transformations [Engstrom et al. (2017)], which showed that l_∞ robust networks are highly vulnerable to adversarial images generated by rotating and translating correctly classified images. Surprisingly, this effect persisted even after augmenting the data with rotated and translated images.



	Model	Natural	Worst-of-10	FO	Grid
MNIST	Standard	99.31%	73.32%	79.84%	**26.02%**
	ℓ_∞-Adversarially Trained	98.65%	51.18%	81.23%	**1.20%**
	Aug. 30 ($\pm$3px, $\pm30°$)	99.53%	98.33%	98.78%	**95.79%**
	Aug. 40 ($\pm4\pm$, $\pm40°$)	99.34%	98.49%	98.74%	**96.95%**
CIFAR10	Standard	92.62%	20.13%	62.69%	**2.80%**
	No Crop	90.34%	15.04%	52.27%	**1.86%**
	ℓ_∞-Adversarially Trained	80.21%	19.38%	33.24%	**6.02%**
	Aug. 30 ($\pm$3px, $\pm30°$)	90.02%	79.92%	85.92%	**58.92%**
	Aug. 40 ($\pm$4px, $\pm40°$)	88.83%	80.47%	85.48%	**61.69%**
ImageNet	Standard	75.96%	47.83%	63.12%	**31.42%**
	No Crop	70.81%	35.52%	55.93%	**16.52%**
	Aug. 30 ($\pm$24px, $\pm30°$)	65.96%	50.62%	66.05%	**32.90%**
	Aug. 40 ($\pm$32px, $\pm40°$)	66.19%	51.11%	66.14%	**33.86%**

Figure 10: Rotation and Translation attacks

Notice that l_∞ robust networks do even worse than standard networks. This result seems to contradict Wong et al's result that l_∞ robustness transfers to Wasserstein attacks.

The paper Robustness May Be at Odds with Accuracy [Tsipras et al. (2018)] pointed out that adversarially robust networks cannot use features that are very weakly correlated with the output, even when that would greatly increase standard accuracy. They are forced to rely on a few robust features, which are more like the ones humans rely on.

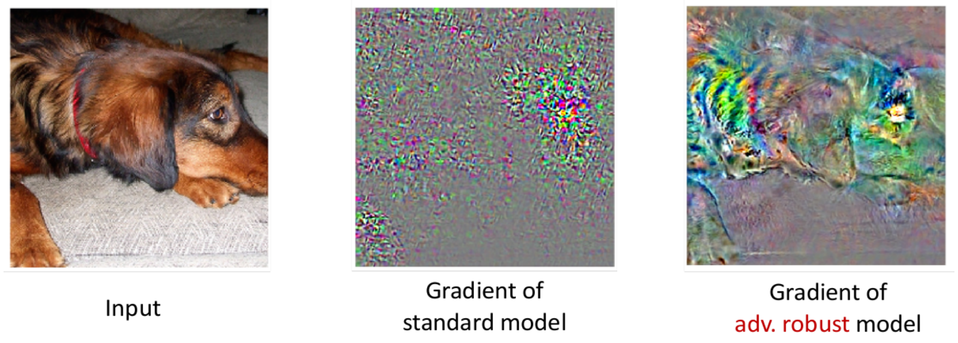


Figure 11: Heat map of gradient activations for each pixel [Madry & Kolter (2018)]

As the figure shows, the gradient of the robust model appears far more human-interpretable. Since human beings are robust to imperceptible perturbations by definition, it is plausible that models using only a few robust features would also be invariant to all such perturbations, explaining transferability of robustness.

5 EXPERIMENTAL WORK

5.1 TRANSFERABILITY AND STABILITY OF THE ATTACK

We performed a wide variety of experiments to study the nature of Adversarial examples, particularly to highlight some of the unexpected properties that they enjoy.

Note that all images in the rest of the report show the perturbed image on the right.

Consider again, the PGD attack on the Hamster image:

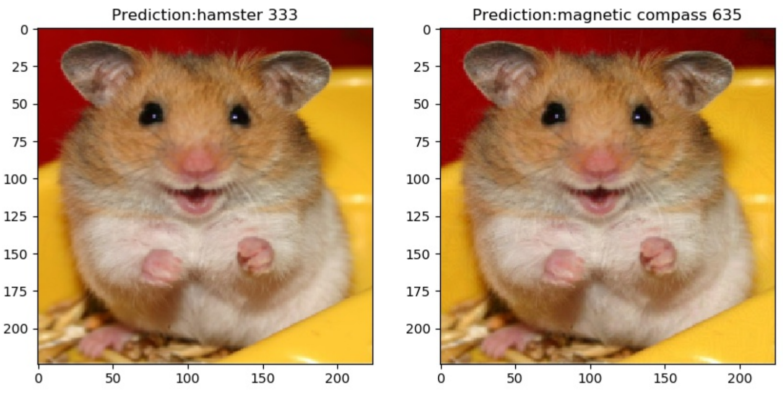


Figure 12: Directed PGD attack to misclassify the Image as a Magnetic Compass with 50 iterations and epsilon = 2/255 on ResNet18. The original image is correctly classified with 99.989% confidence, while the attacked image is misclassified with 99.97% confidence

The Perturbation δ is almost imperceptible to the human eye, and must be magnified by several times to appreciate the underlying structure. The first image was modified to have mean $= 127$ across all three channels, else it would appear completely black. A few pixel values had to be clipped to be within range for the second image, leading to a small distortion in colour space. We will, for comparison purposes, utilise this same perturbation for all the following experiments as well.

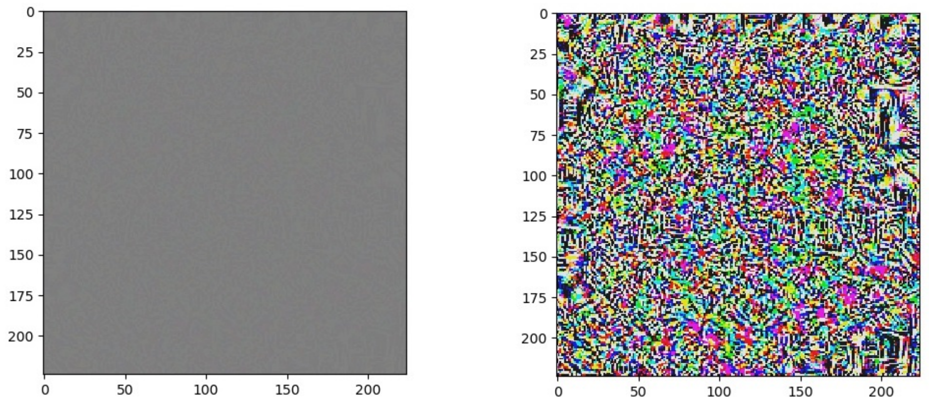


Figure 13: Perturbation under 1X and 50X Magnification

Next, we looked at the stability of the Adversarial perturbation to the addition of random noise to the image. We find that the attack is very stable, and noise sampled from a uniform distribution that is large relative to the perturbation itself, still does not avert a misclassification.

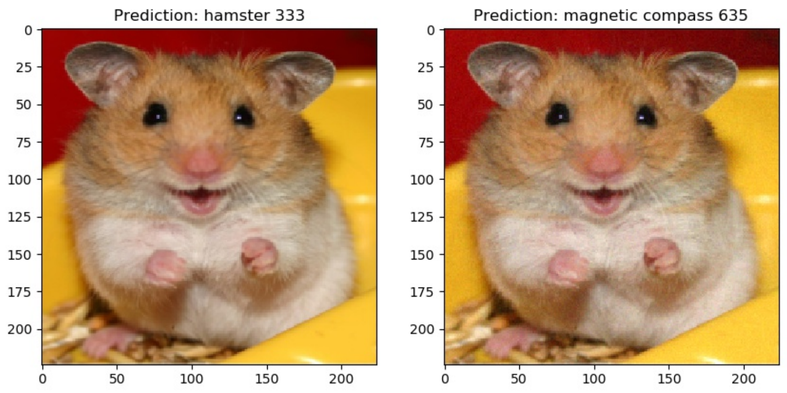


Figure 14: Identical Perturbation with 7X Noise (uniform in $[0, 1] \cdot 7\epsilon$) added still fools ResNet18, causing a wrong prediction with 94.2% confidence

We further wanted to see the effect of the perturbation on images which seem not too different to the human eye, but very different individual pixel values. Thus we applied the same perturbation to a flipped image of the hamster. We found that we had to increase the value of epsilon (14.0/255) to successfully fool the network.

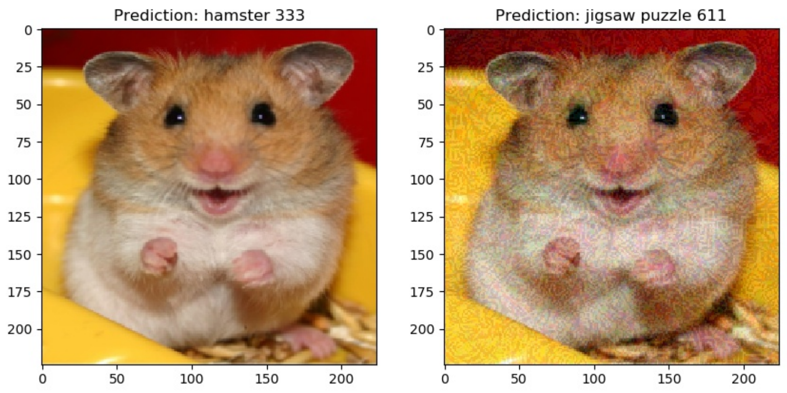


Figure 15: A Very Suprising Result: Though the flipped image of the Hamster is correctly identified with more than 99% accuracy, and Jigsaw-Puzzle having predicted probability very close to zero, the same perturbation (epsilon = 2/255) appears to still cause a cross-over in the decision boundaries. However, here the confidence for Jig-saw Puzzle for the perturbed image is lower-only 74%, as this was not a targeted attack to misclassify as a Jigsaw-Puzzle, but instead resembles an undirected attack

To study a little more about the nature of the perturbation, we looked at its efficacy on a completely different data-distribution. We chose a particularly interesting example where the ResNet initially misclassified the original image.

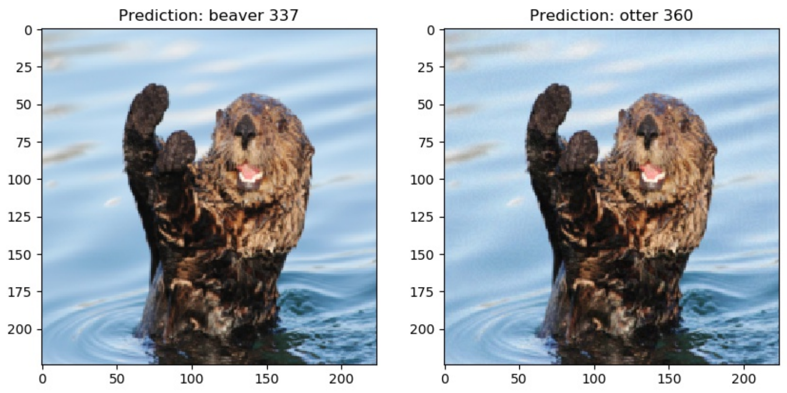


Figure 16: Upon adding the original delta that caused the Hamster image to be predicted as a Magnetic compass, we again see a cross-over between decision boundaries: The original probabilities for otter and beaver was 33% and 64% respectively, but for the second image the predictions switch: they become 52% and 46% respectively. This indicates that the perturbation is far from being random noise even for seemingly disparate classes.

To further study the robustness of the adversarial attack, we intended to add both systemic and random noise to the image. To do so, inspired from 3D-printed adversarial examples created by Athalye et al. (2017), we took a picture of the laptop screen with the adversarial image with our mobile phone, and then ran this on the network. This gave the first negative result amongst all the experiments, which is to say, the network was not fooled.

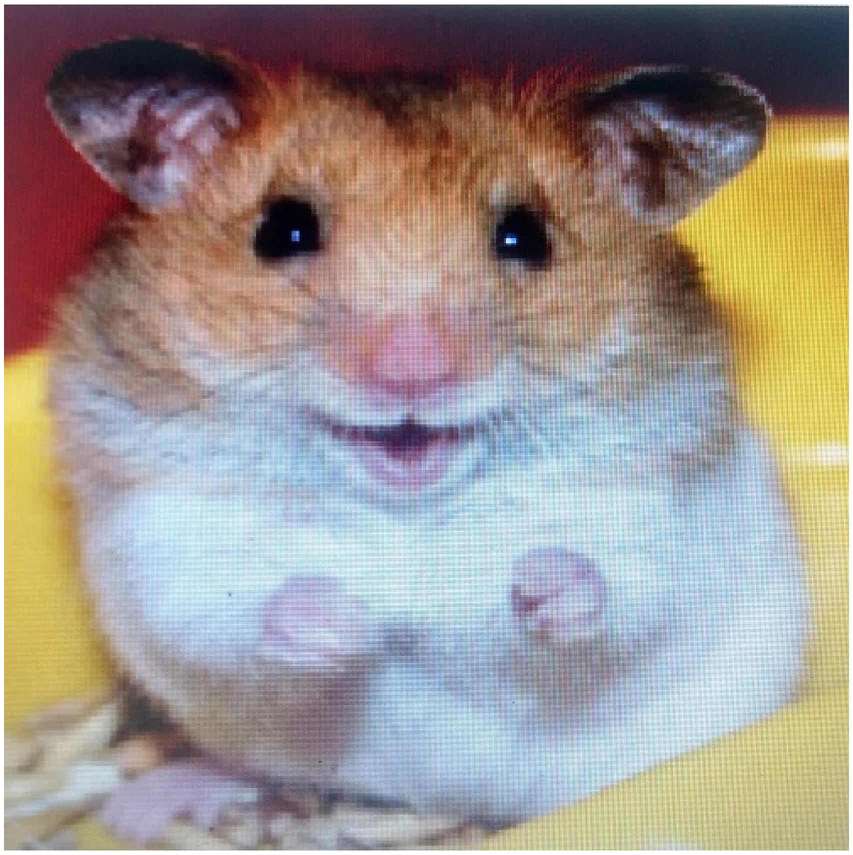


Figure 17: Very Severe Distortions can be seen in the Mobile-Camera Picture of the Perturbed Image, leading to nullification of the Attack

Given a fixed epsilon, it was found that ADAM was a much better optimiser than SGD. We thus made a plot for the Directed PGD Attack.

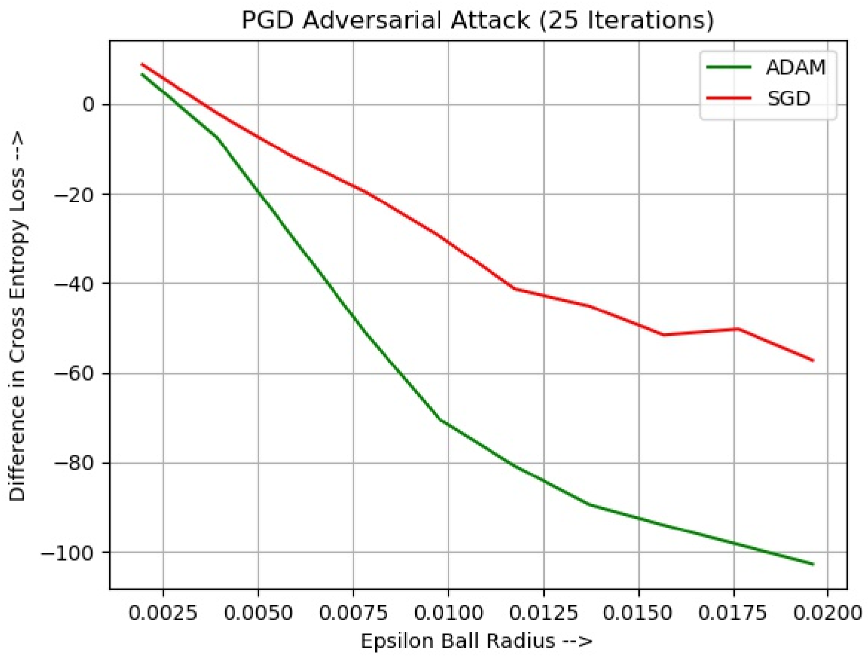


Figure 18: ADAM was initialised with learning rate 0.001, $(\beta_1, \beta_2) = (0.9, 0.999)$ and $\epsilon = 10^{-8}$. SGD was initialised with learning rate 0.07. All experiments were carried out on the same Hamster Image for comparison purposes.

5.2 Transferability of Robustness

Now, in pursuit of learning about the transferability of robustness to different kinds of Adversarial attacks, we looked at the robustness of a network adversarially trained on Wasserstein examples. For this, we used the network trained on the standard MNIST dataset as well as Wasserstein adversarial images as uploaded on the [Wong et al. (2019)] Github page. We studied the robustness of this network(which had 2 convolutional layers, and 2 fully-connected layers, with ReLU activations) to l_∞ attacks, as well as adversarial attacks generated by composing rotations and translations as in the [Engstrom et al. (2017)] paper.

For comparison purposes, we perform the same experiments with a network with identical architecture, but which has not been exposed to Wasserstein attacks during training time and only to standard MNIST training examples, which we refer to as the Vanilla Network in the following sections.

We decided to work with MNIST images as they were easier to visualise and interpret the perturbations, as compared to CIFAR10. Ideally, we would have liked to have worked with a Deep Network adversarially trained on ImageNet, but no such network is yet publicly available, and training such a network that simultaneously requires generation of a large sample of Wasserstein examples is beyond the compute power available to us.

We study the behaviour of the two networks, as we intuitively expect small rotations and translations to have small Wasserstein distance from the original image.

Standard Accuracy: The Vanilla network actually slightly out-performs the Wasserstein-Adversarially network on the Test Set of MNIST: 98.90% for the former and 96.95% for the latter network.

l_∞ **Attacks:** We ran Directed Projected Gradient Descent Attacks on unseen MNIST Test images, to try and make the network misclassify any input image as a '5'. We found some interesting results in both networks:
In the Vanilla Network, we found that the network was fairly robust to l_∞ attacks with epsilon $\leq 12/255$, having accuracy over 95%. Furthermore, we were surprised when we discovered a sharp "epsilon-cliff" on a network that was not adversarially trained. It was also observed that the directed PGD attack for any given digit does not transfer well to other images, even some other with the same label. This transferability seems work better with larger images like 224x224x3 as in ImageNet.

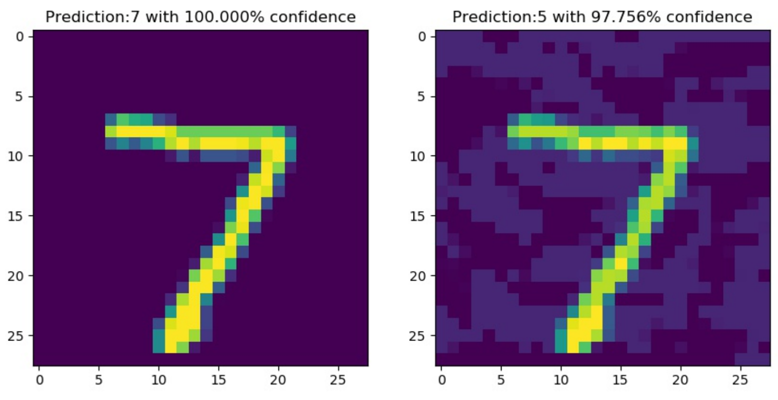


Figure 19: Directed PGD l_∞ Attack on Vanilla network with 100 iterations using ADAM, epsilon = 15/255

ϵ	12/255	13/255	14/255	15/255	16/255
Prediction	7	7	5	5	5
Prediction Accuracy	97.16%	76.24%	79.44%	97.75%	99.64%

Table 1: Epsilon-cliff in directed PGD l_∞ attack to turn a 100% 7 into a 5

For the Wasserstein-Adversarially trained net, it was found that it was *extremely robust* against l_∞ attacks, with the accuracy falling below 90% only for relatively large epsilon values, over 40/255.

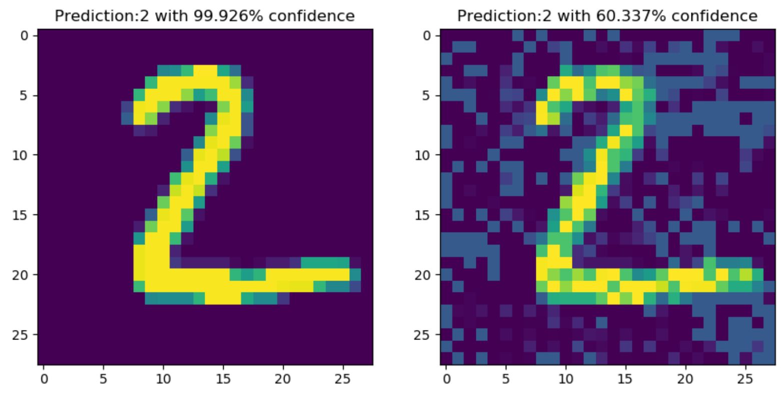


Figure 20: Directed PGD l_∞ Attack on Wasserstein network with 100 iterations using ADAM, epsilon = 40/255. It was observed for several examples that though the confidence was lowered, it rarely fell below 50%, indicating correct predictions

Adversarial Attacks with Only Rotation and Translation: As detailed in the [Engstrom et al. (2017)] paper, it suffices to only use simple rotations and translations to fool fairly sophisticated convolutional networks. The paper lays down several methods to produce adversarial attacks, which include Worst-of-k Attacks, Random Search and Grid Search in the parameter space for rotation, horizontal and vertical translations. In the paper, they report that the best results were found while performing exhaustive grid-search.

We decided to use grid-search for producing the attacks. However, we limited the range of rotation of images to be within 20-degrees clockwise and anticlockwise in steps of 0.5 degrees, compared to 30 degrees used in the paper (30 degree rotations on MNIST are barely recognizable). We however follow the paper in the translation range: about 10% of the image dimensions, which is 3 pixels for MNIST (being 28x28 images), which was varied in steps of 1 pixel each.

We tried to find the optimal combination of image rotation angle, horizontal and vertical translation that minimises the confidence of the correct label, making it commit an error. We found that in general, these attacks were quite effective on both the Vanilla Network as well as the Wasserstein-Adversarially trained net, although the latter seemed to perform better in general. Not a single example was found where the Vanilla network correctly classified a rotation-translation adversarial image on which the other network committed an error.

Rotation, Translation	Wasserstein trained	Vanilla network
$20°, +-3$ pix	32%	25%
$10°, +-3$ pix	42%	32%
$5°, +-1$ pix	90%	83%

Table 2: Accuracy on rotated translated image: strongest attack chosen by grid search

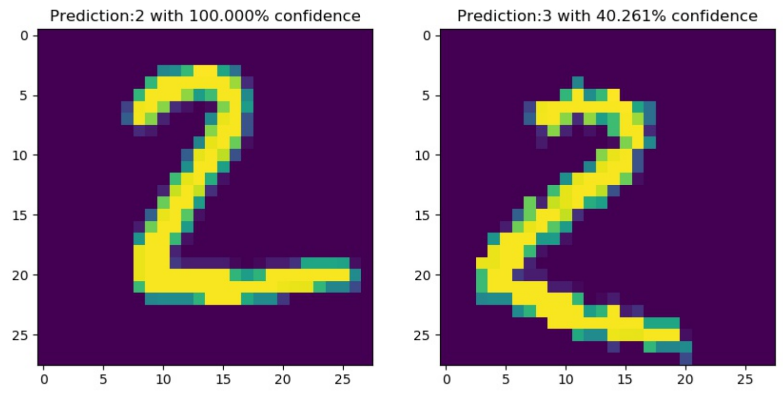

Prediction:2 with 100.000% confidence
Prediction:3 with 40.261% confidence

Figure 21: Adversarial Attack by Only Composing Rotations and Translations. The Vanilla Network predicts the wrong class with fair confidence. 20 degrees clockwise rotation, with 3 pixels left translation and 2 pixels translation upward gave the most effective attack.

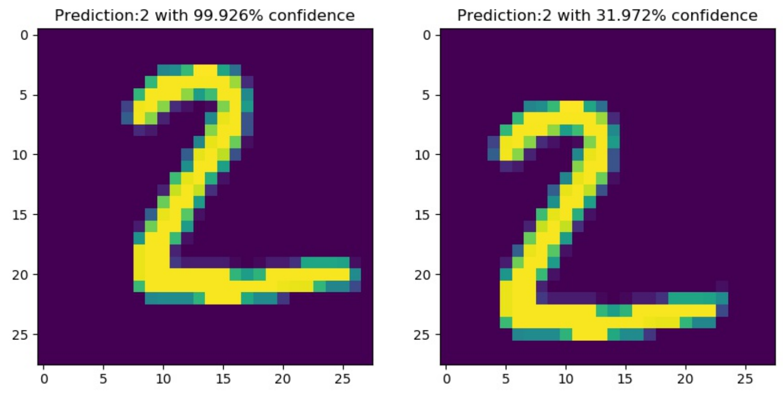

Prediction:2 with 99.926% confidence
Prediction:2 with 31.972% confidence
0
5
10
15
20
25

Figure 22: Adversarial Attack by Only Composing Rotations and Translations. The Wasserstein Adversarially Trained Network predicts the correct class albeit with much lower confidence. 2.5 degrees anticlockwise rotation, with 3 pixels left translation and 3 pixels translation upward gave the most effective attack.

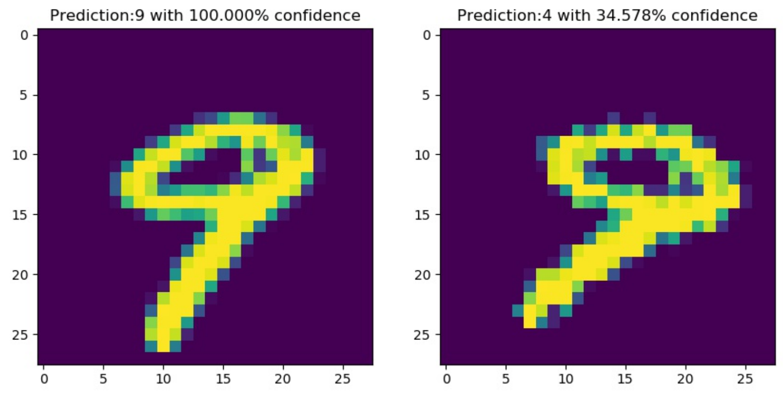

Prediction:9 with 100.000% confidence
Prediction:4 with 34.578% confidence
0
5
10
15
20
25

Figure 23: Adversarial Attack by Only Composing Rotations and Translations. (Vanilla) It was generally observed that for digits that were already a bit rotated, the optimal attack just further rotates the image, along with small translations. The optimal attack as shown here is with 20 degrees clockwise rotation, and 1 pixel right translation .Surprisingly, it was observed even with rotationally symmetric digits like 0.

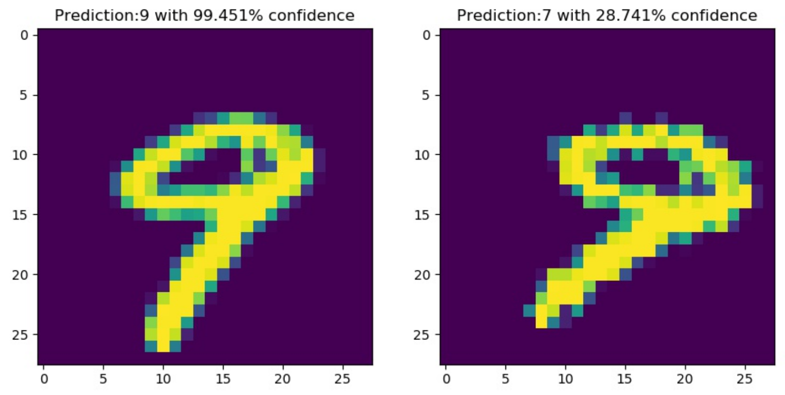


Figure 24: Adversarial Attack by Only Composing Rotations and Translations. The Wasserstein-Robust net in general also had lower confidence while making wrong predictions compared to the first network. Here too a similiar attack turned out to be optimal: 20 degree clockwise rotation, with 2 pixels translation to the right.

6 Conclusions

Our first set of experiments showed that CNNs are sufficiently linear that an adversarial delta designed to cross a decision boundary in one location will in fact work in other locations also. Such a delta is also stable under the addition of large amounts of noise.

We found a sharp epsilon cliff in the Vanilla network for MNIST, but did not find a cliff for ResNet on ImageNet. Such cliffs are often seen in adversarially trained networks, precisely at the epsilon value used in training, but we cannot completely explain why this should happen for a vanilla network.

Our transferability of robustness experiments showed that Wasserstein robustness transfers extremely well to l_∞ attacks, which is quite surprising, since these can be extremely large in the Wasserstein norm.

A possible explanation is that adversarially trained networks are forced to rely on fewer features that are highly correlated with the output. As pointed out in the Transferability of Robustness section, this could make them less vulnerable to all perturbations of brittle, weakly-correlated features.

6.1 The intriguing Trilemma

In our presentation we pointed out that the following three results appeared to be mutually contradictory:

1. Wong et al. (2019) - l_∞ robustness provided substantial protection against Wasserstein attacks
2. Engstrom et al. (2017) - l_∞ robustness is completely orthogonal to rotation-plus-translation attacks
3. Rotations and translations are relatively small in the Wasserstein norm

We came up with some ideas to explain this discrepancy. While rotations and translations are intuitively small in the Wasserstein metric, Wong et al used a maximum epsilon of 2.1 in training their adversarially robust network. However a simple one pixel translation would have a distance of 1 unit. So rotations and translations are not as small as what one might expect at first glance. Another point is that Wong et al only used local 5 x 5 transport plans, for computational reasons. This is quite a restrictive subset of the Wasserstein ball, and does not include translations and rotations. As pointed out in the Wasserstein Results section, Wong et al did not attack their model down to zero percent accuracy. This could be due to Sinkhorn iteration being unable to find a point close enough to the optimum. Our experimental results on rotations show that Wasserstein adversarial training provides some amount of robustness to rotations and translations, but not much compared to its robustness to l_∞ attacks.

Since l_∞-trained networks do very poorly on rotations and translations, Wasserstein training offers a good balance of robustness to a variety of attacks that reflect our intuition of visual similarity.

References

Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pp. 2292–2300, 2013.

Logan Engstrom, Brandon Tran, Dimitris Tsipras, Ludwig Schmidt, and Aleksander Madry. A rotation and a translation suffice: Fooling cnns with simple transformations. *arXiv preprint arXiv:1712.02779*, 2017.

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

Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

Alexsandr Madry and Zico Kolter. Adversarial robustness; theory and practice, 2018. URL `https://adversarial-ml-tutorial.org/`. NeurIPS tutorial on Adversarial Robustness.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *stat*, 1050:11, 2018.

Cédric Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.

Eric Wong, Frank R Schmidt, and J Zico Kolter. Wasserstein adversarial examples via projected sinkhorn iterations. *arXiv preprint arXiv:1902.07906*, 2019.

7 Network Architecture

For the Vanilla and Wasserstein trained networks:

```
net = nn.Sequential(
    nn.Conv2d(1, 16, 4, stride=2, padding=1),
    nn.ReLU(),
    nn.Conv2d(16, 32, 4, stride=2, padding=1),
    nn.ReLU(),
    Flatten(),
    nn.Linear(32*7*7,100),
    nn.ReLU(),
    nn.Linear(100, 10) )
```