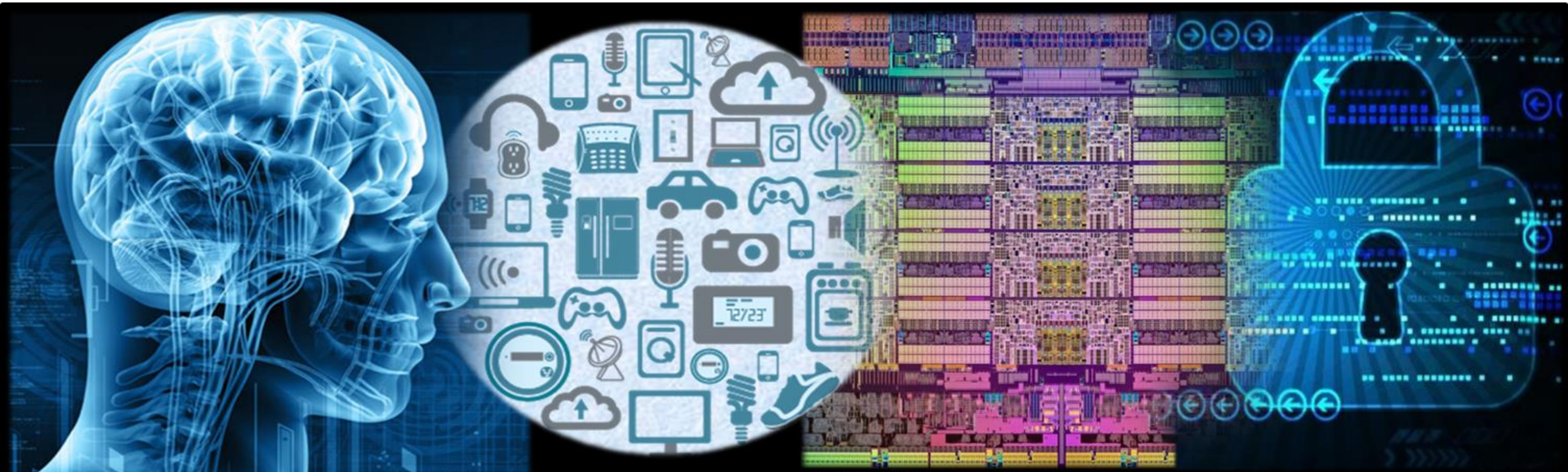
Progress Report

Ahmad Savaiz Nazir
eBrain Lab, New York University (NYU) Abu Dhabi, UAE



Clean Label Backdoor Attack Method 2

- CIFAR-10 with ResNet18 model implementation complete with accuracy up to 97%
- Adversarial perturbation trigger generation and visual verification complete
- Iteratively crafted trigger for stronger model
- Triggers embedded into training dataset but insufficient testing

Problems:

- Baseline accuracy drops on test dataset suggesting over fitting
- Attack success rate is still near zero

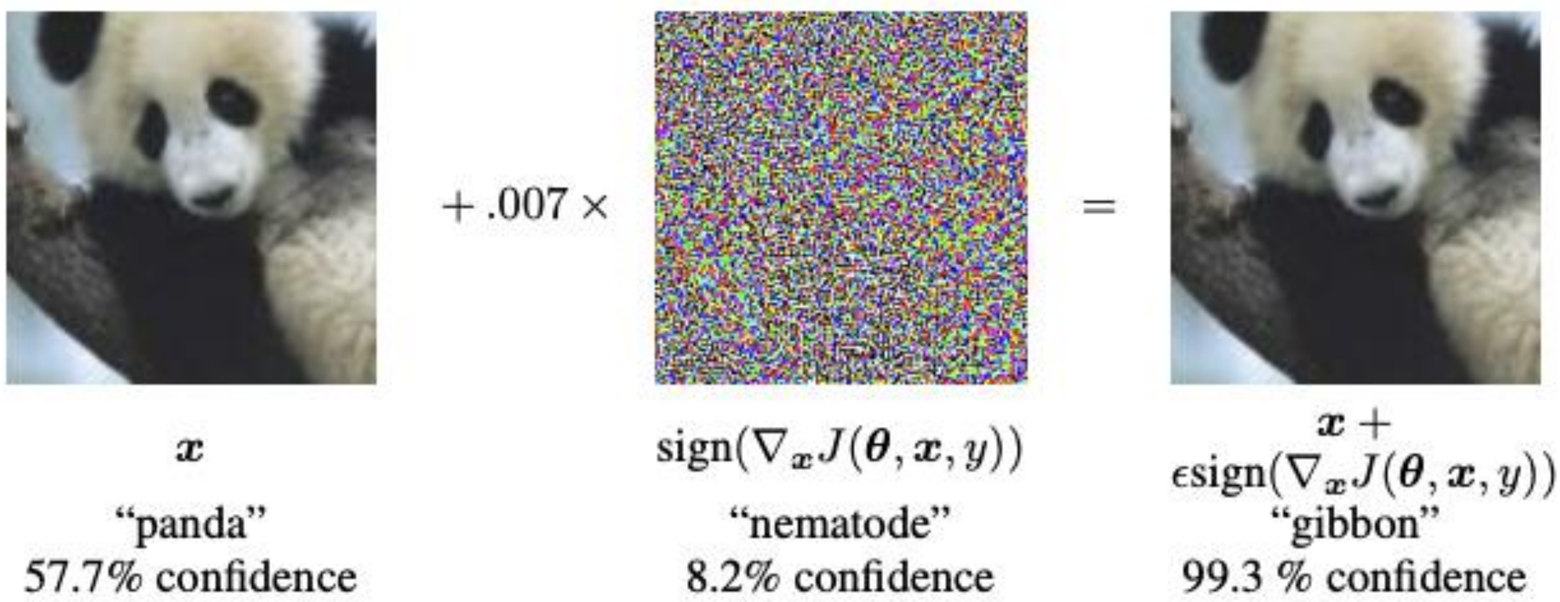
Tried:

- Changing optimizer and incorporating learning rate scheduler
- Tested with various epsilons
- Data augmentation
- Different models from ResNet18

FGSM

Fast Gradient Sign Method

- The fast gradient sign method works by using the gradients of the neural network to create an adversarial example. For an input image, the method uses the gradients of the loss with respect to the input image to create a new image that maximizes the loss. This new image is called the adversarial image.



From the Paper

4.3 Method 2: Adversarial perturbations

Adversarial examples are natural inputs that have been slightly perturbed with the goal of being misclassified by an ML model (Section 4.3). In fact, the perturbations have been found to transfer across different models or even across different architectures (Szegedy et al., 2014; Papernot et al., 2016).

Here, we utilize adversarial perturbations and their transferability across models and architectures in a somewhat unusual way. Instead of causing a model to misclassify an input during inference, we use them to cause the model to misclassify during *training*. Specifically, we will apply an adversarial transformation to the poisoned inputs (before applying the trigger) to make them harder to learn. Concretely, given a pre-trained classifier f_θ with loss $\mathcal{L}$ and an input-label pair (x, y), we construct a perturbed variant of x as

$$x_{\text{adv}} = \underset{\|x'-x\|_p \leq \varepsilon}{\arg\max}\ \mathcal{L}(x', y, \theta),$$

for some ℓ_p-norm and a small constant ε. We solve this optimization problem using a standard method in this context, projected gradient descent (PGD) (Madry et al., 2018) (see Appendix A for more details). In fact, we will use perturbations based on adversarially trained models (Madry et al., 2018) since these perturbations are more likely to resemble the target class for large ε (Tsipras et al., 2019).

We use x_{adv} along with the original, ground-truth label of x (the target label) as our poisoned input-label pair. By controlling the value of ε we can vary the resulting image from slightly perturbed to visibly incorrect (Figure 5). Note that these adversarial examples are computed with respect to an independent, pre-trained model since the adversary does not have access to the training process.

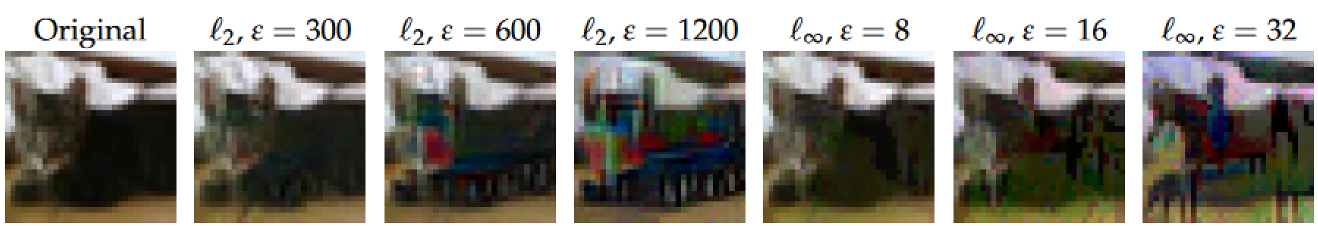
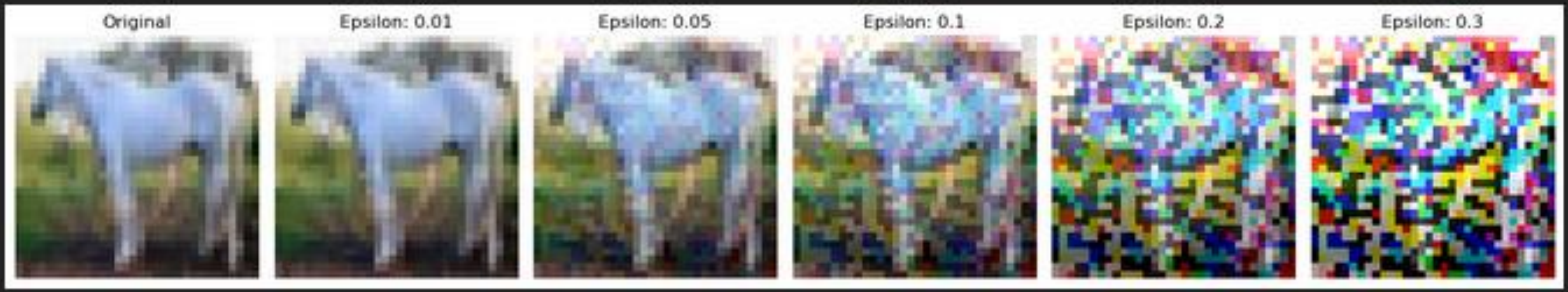


Figure 5: Example of adversarial perturbations for different levels of distortion (ε) bounded in ℓ_2- and ℓ_∞-norm for adversarially trained models (pixels lie in $[0, 255]$). Additional examples in Appendix Figure 21.

Example with Different Epsilon



Code

```python
def craft_trigger(model, x_source, y_target, epsilon, iterations=10):
    """
    Iteratively crafts a trigger using adversarial training.
    """
    x_trigger = torch.rand_like(x_source[:, :, :3, :3])
    x_trigger = x_trigger.to(device).detach().requires_grad_(True)

    for _ in range(iterations):
        x_sample_with_trigger = x_source.clone()
        x_sample_with_trigger[:, :, :3, :3] = x_trigger

        outputs = model(x_sample_with_trigger)
        loss = criterion(outputs, y_target)
        model.zero_grad()
        loss.backward()

        data_grad = x_trigger.grad.data
        x_trigger = x_trigger + epsilon * torch.sign(data_grad)
        x_trigger = torch.clamp(x_trigger, 0, 1)
        x_trigger.detach_()
        x_trigger.requires_grad_(True)

    return x_trigger

x_source, _ = next(iter(train_loader))
x_source = x_source[:8].to(device)
y_target = torch.tensor([5] * x_source.size(0), device=device)

epsilon = 0.15 # Adjusted epsilon
triggers = craft_trigger(model, x_source, y_target, epsilon)
trigger = triggers[0:1]
```

```python
def iteratively_craft_trigger(model, x_source, y_target, epsilon, iterations=10):
    """
    Iteratively crafts a trigger using adversarial training.
    """
    x_trigger = x_source.clone().detach().requires_grad_(True)
    for _ in range(iterations):
        outputs = model(x_trigger)
        loss = criterion(outputs, y_target)
        model.zero_grad()
        loss.backward()
        data_grad = x_trigger.grad.data
        x_trigger = x_trigger + epsilon * torch.sign(data_grad)
        x_trigger = torch.clamp(x_trigger, 0, 1)
        x_trigger.detach_()
        x_trigger.requires_grad_(True)
    return x_trigger

def test_with_full_trigger(model, test_loader, trigger):
    model.eval()
    correct = 0
    for images, _ in test_loader:
        images = images.to(device)
        images[:] = trigger
        outputs = model(images)
        pred = outputs.argmax(dim=1)
        correct += (pred == y_target[0]).sum().item()
    return 100. * correct / len(test_loader.dataset)
```

Next Steps

- Writing a new implementation with different infrastructure
- Current progress – fixing bugs

Thank You!

Ahmad Savaiz Nazir

an3909@nyu.edu