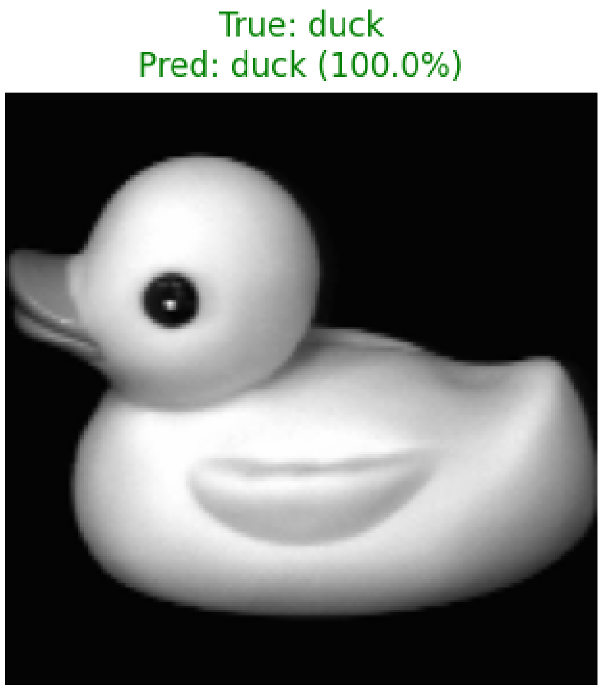
**Adversarial attack generation**

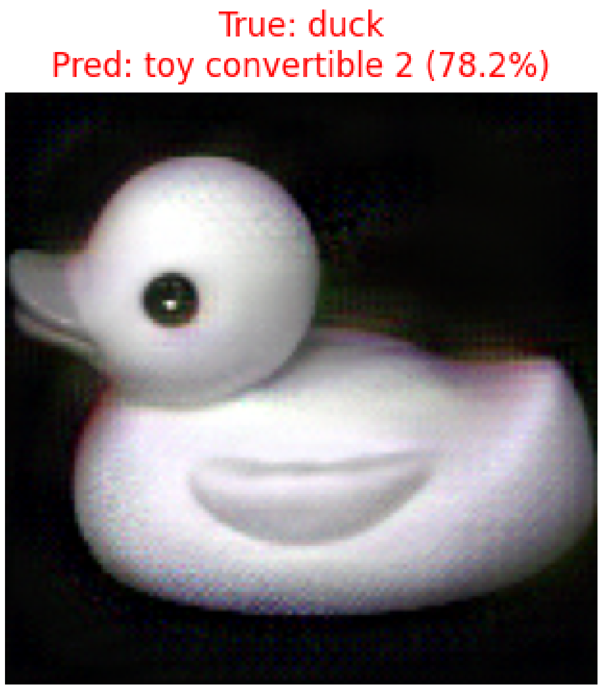
We trained the network with a loss function that we can call $J(\theta, x, y)$, performing gradient descent with respect to the parameters $\theta$. We can just as easily, especially with PyTorch, compute the gradient with respect to the input $x$. If we allow our input to be an image from the training data, which results in a correct output with nearly 100% accuracy, we can perform gradient *ascent* with the goal of finding the most similar issue that is wrongly detected as a different item. This can be represented as follows:

$$x_{\text{step}} = x + \delta \nabla_x J(\theta, x, y)$$

To see this in action, we will apply it to the first image from the dataset:



After applying the attack, we get the following image and prediction.



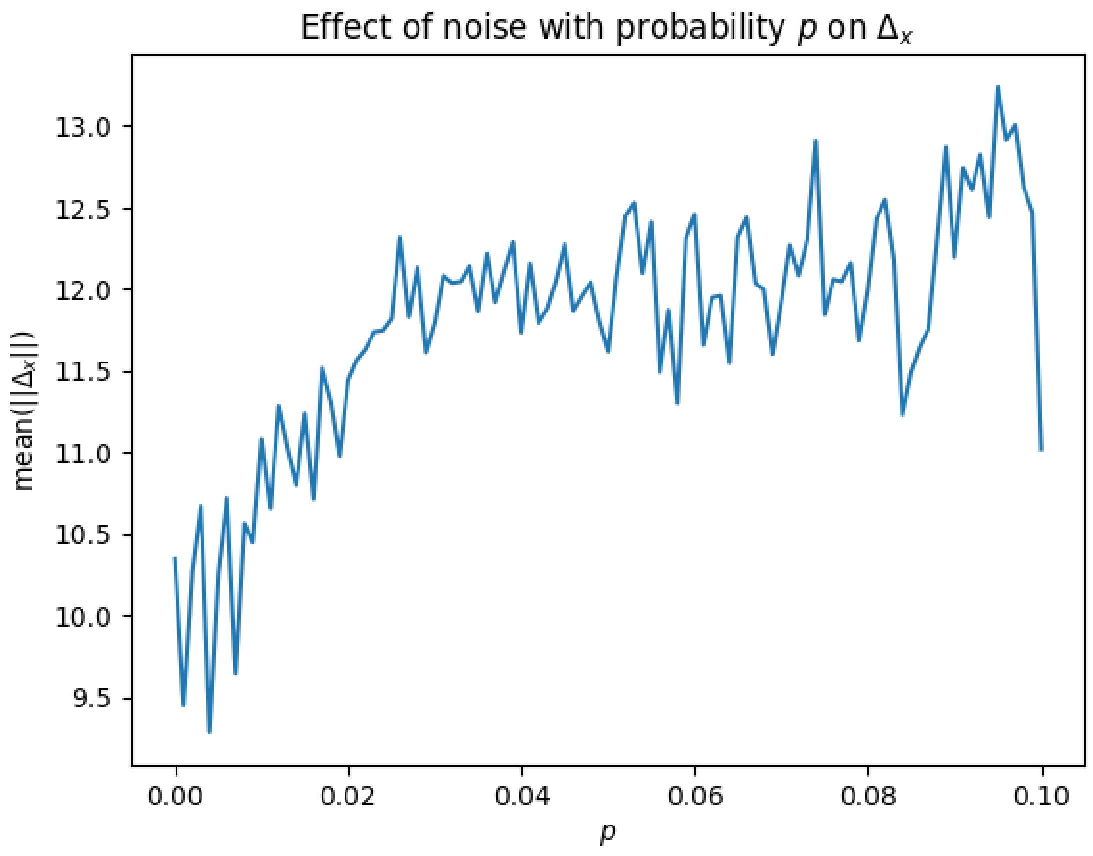
To the human eye, the images look almost identical, however the neural network was completely tricked. The obvious followup question is: how can we make the neural network more resistant to this attack?

Since the changes to the image look a lot like just adding some noise, we will do exactly that during training.

To measure the resistance of a neural network against this attack, we will comput the following using each item in the dataset in the attack:

$$\Delta_x = x_{false} - x_{\text{original}}$$

This is the elementwise difference between the false image and the original. By computing the norm of this value over the whole dataset for neural networks with different noise probabilities, we can measure the effect of noise in producing a more robust neural network:



In the above plot, we can see that the neural network becomes more robust against these attacks by adding noise to the training data.