Vulnerability of machine learning models to adversarial examples

Petra Vidnerová

Institute of Computer Science
The Czech Academy of Sciences

Hora Informaticae 2016

Outline

- Introduction
- Works on adversarial examples
- Our work
  - Genetic algorithm
  - Experiments on MNIST
- Ways to robustness to adversarial examples

Introduction

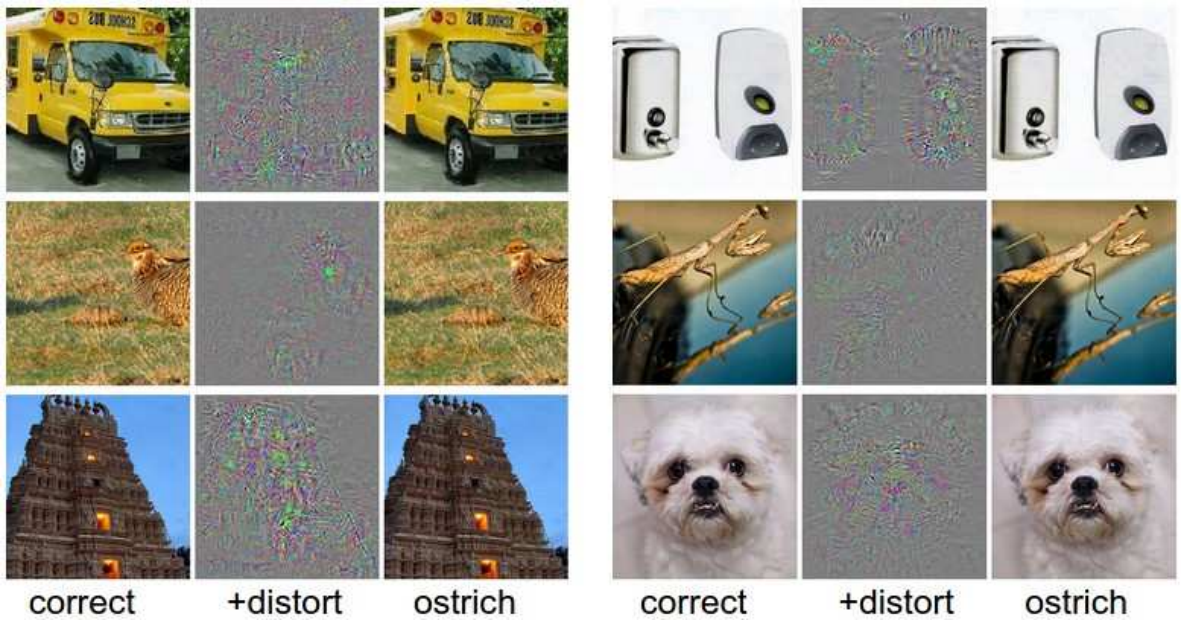
- Applying an imperceptible non-random perturbation to an input image, it is possible to arbitrarily change the machine learning model prediction.

Figure from *Explaining and Harnessing Adversarial Examples* by Goodfellow et al.

- Such perturbed examples are known as *adversarial examples*. For human eye, they seem close to the original examples.
- They represent a security flaw in classifier.

Works on adversarial examples I.

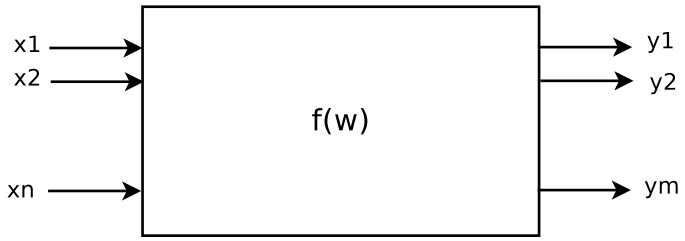
- *Intriguing properties of neural networks*. 2014,Christian Szegedy et al.



- Perturbations are found by optimising the input to maximize the prediction error (L-BFGS).

Works on adversarial examples I.

Learning



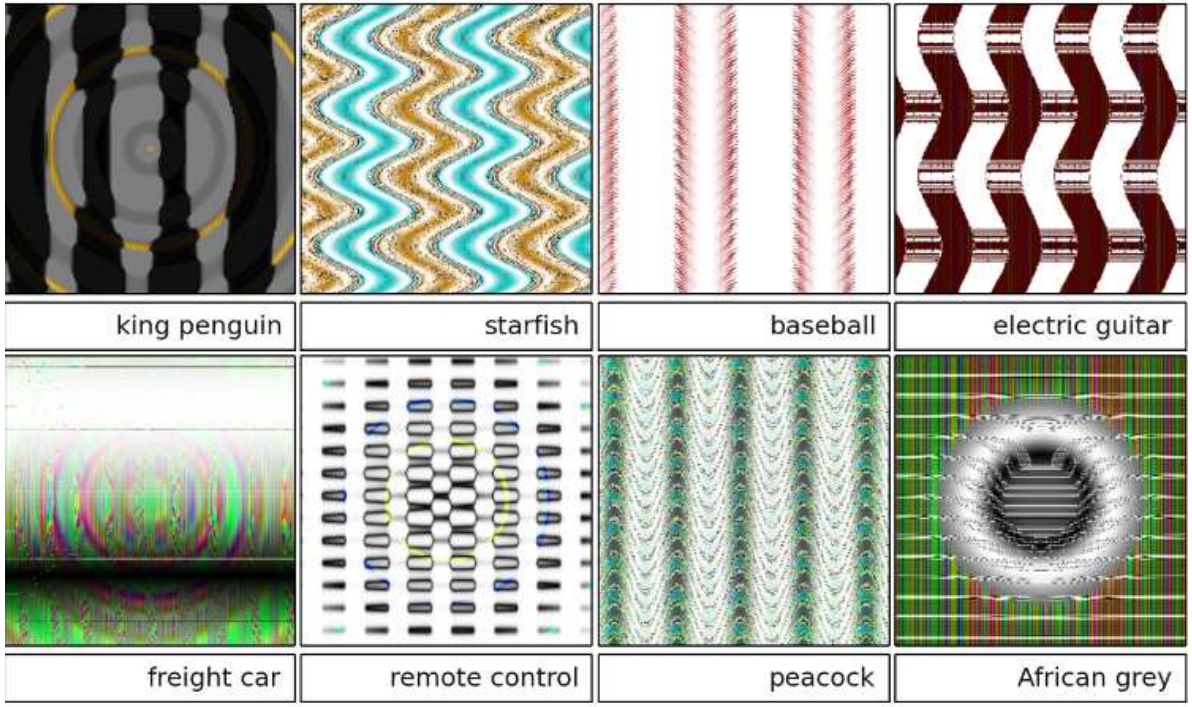
- model $f_{\vec{w}} : \mathbb{R}^n \to \mathbb{R}^m$
- error func.: $E(\vec{w}) = \sum_{i=1}^{N} e(f_{\vec{w}}(x_i), y_i) = \sum_{i=1}^{N} (f_{\vec{w}}(x_i) - y_i)^2$
- learning: $\min_{\vec{w}} E(\vec{w})$

Finding adversarial example

- $\vec{w}$ is fixed, $\vec{x}$ is optimized
- minimize $||r||_2$ subject to $f(x + r) = l$ and $(x + r) \in [0, 1]^m$
- a box-constrained L-BFGS

Works on adversarial examples II.

- *Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images*
  2015,Anh Nguyen, Jason Yosinski, Jeff Clune

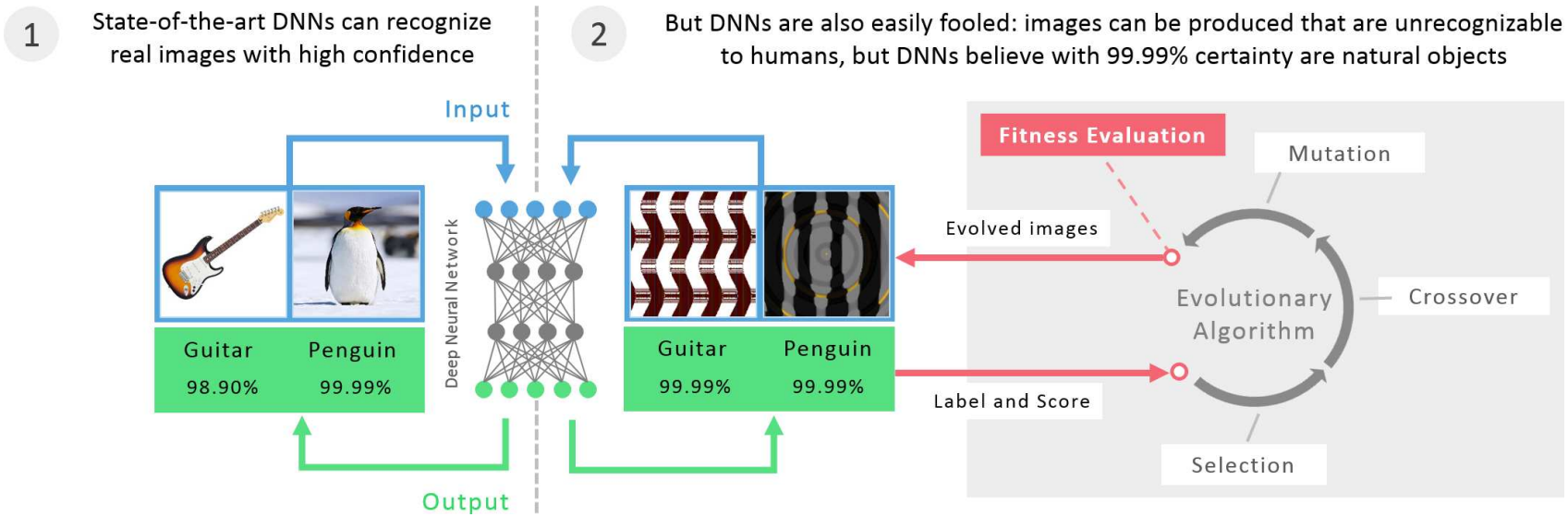


- evolutionary generated images

Works on adversarial examples II.

Compositional pattern-producing network (CPPN)

- similar structure to neural networks
- takes (x, y) as an input, outputs pixel value
- nodes: sin, sigmoid, Gaussian, and linear



Works on adversarial examples III.

- *Explaining and Harnessing Adversarial Examples* 2015,Goodfellow et al.
- linear behaviour in high dimensional spaces is sufficent to cause adversarial examples

$$\tilde{x} = x + \eta$$

$x, \tilde{x}$ belong to the same class if $||\eta||_\infty < \epsilon$

$$w^T \tilde{x} = w^T x + w^T \eta$$

for $\eta = \epsilon sign(w)$ activation increases ϵmn
$||\eta||_\infty$ does not grow with dimensionality, but ϵmn does

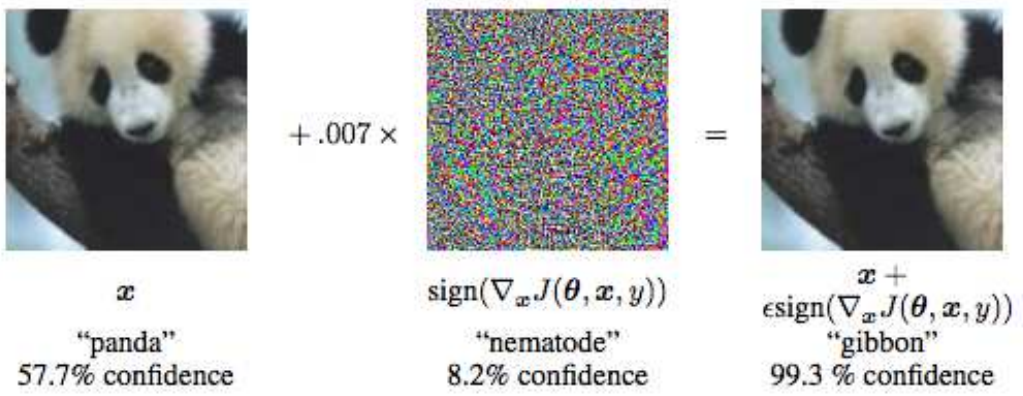
- in large dimensions small changes of the input cause large change to the output

Works on adversarial examples III.

- nonlinear models: parameters θ, input x, target y, cost function $J(\theta, x, y)$
- we can linearize the cost function around θ and obtain optimal perturbation

$$\eta = \epsilon sign(\nabla_x J(\theta, x, y))$$

- adding small vector in the direction of the sign of the derivation – *fast gradient sign method*



Our work

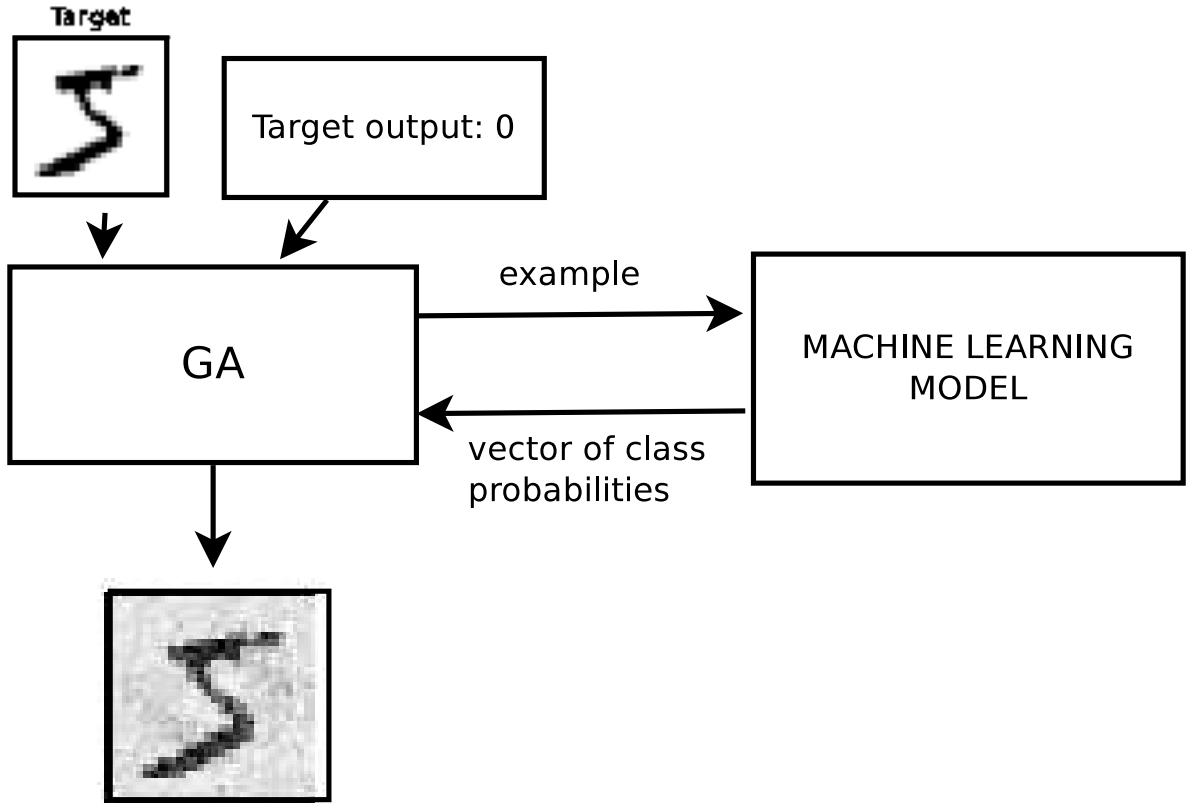
- genetic algorihms used to search for adversarial examples
- tested various machine learning models including both deep and shallow architectures

Search for adversarial images

- To obtain an adversarial example for the trained machine learning model, we need to *optimize the input image with respect to model output*.
- For this task we employ a GA – robust optimisation method working with the whole population of feasible solutions.
- The population evolves using operators of selection, mutation, and crossover.
- The machine learning model and the target output are fixed.

Black box approach

- genetic algorithms to generate adversarial examples
- machine learning method is a blackbox
- applicable to all methods without the need to acess models parameters (weights)



Genetic algorithm

- *Individual:* image encoded as a vector of pixel values:

$$I = \{i_1, i_2, \ldots, i_N\},$$

  where $i_i \in < 0, 1 >$ are levels of grey and N is a size of flatten image.
- *Crossover:* operator performs a two-point crossover.
- *Mutation:* with the probability p_{mutate_pixel} each pixel is changed:

$$i_i = i_i + r,$$

  where r is drawn from Gaussian distribution.
- *Selection:* $3-$tournament

GA fitness

- The fitness function should reflect the following two criteria:
  - the individual should resemble the target image
  - if we evaluate the individual by our machine learning model, we would like to obtain a target output (i.e. misclassify it).

Thus, in our case, a fitness function is defined as:

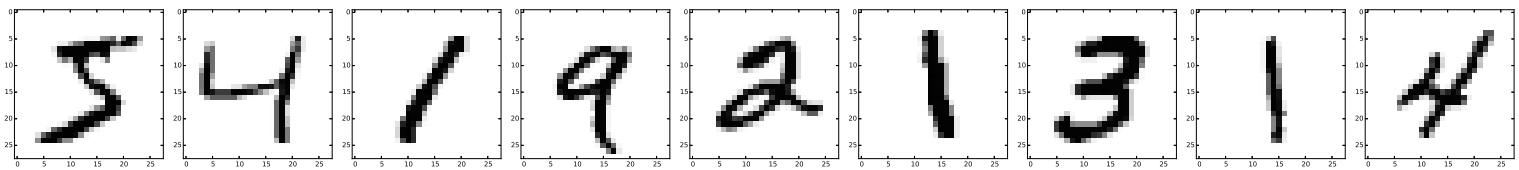
$$f(I) = -(\quad 0.5 * cdist(I, target\_image) \quad (1)$$
$$+ \quad 0.5 * cdist(model(I), target\_answer)), \quad (2)$$

where *cdist* is an Euclidean distance.

Dataset for our experiments

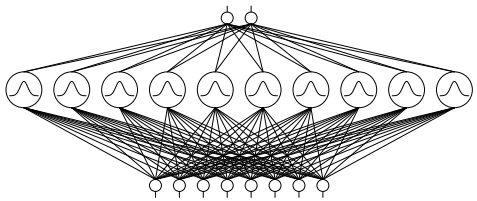
MNIST dataset

- 70000 images of handwritten digits
- 28×28 pixels
- 60000 for training, 10000 for testing



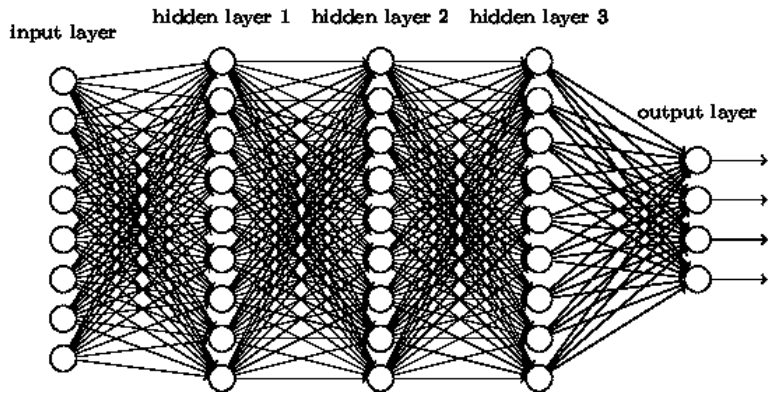
Machine learning models overview

- Shallow architectures
  - SVM — support vector machine
  - RBF — RBF network



- DT — decision tree

- Deep architectures
  - MLP — multilayer perceptron network
  - CNN — convolutional network



Support Vector Machines (SVM)

- popular kernel method
- learning based on searching for a separating hyperplane with highest margin
- one hidden layer of kernel units, linear output layer

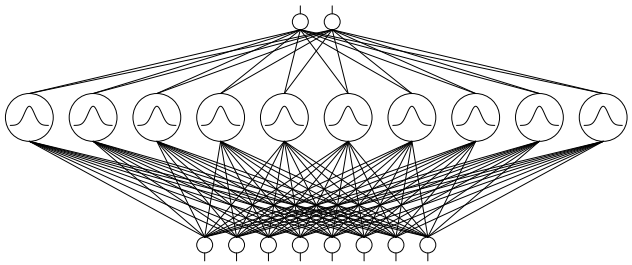
Kernels used in experiments:

- linear $\langle x, x' \rangle$
- polynomial $(\gamma \langle x, x' \rangle + r)^d$, grade 2 and 4
- Gaussian $\exp(-\gamma |x - x'|^2)$
- sigmoid $\tanh(\gamma \langle x, x' \rangle + r)$.

Implementation: SCIKIT-learn library

RBF network

- feedforward network with one hidden layer, linear output layer
- local units (typically Gaussian functions)



- our own implementation
- 1000 Gaussian units

Decision Tree (DT)

- a non-parametric supervised learning method

Implementation: SCIKIT-learn

Deep neural networks

- feedforward neural networks with multiple hidden layers between the input and output layer

Multilayer perceptrons (MLP)

- Perceptron units with *sigmoid* function
- Rectified linear unit (ReLU): $y(z) = max(0, z)$.

Implementation:

- KERAS library
- MLP — three fully connected layers, two hidden layers have 512 ReLUs each, using dropout; the output layer has 10 softmax units.

Convolutional Networks (CNN)

- *Convolutional units* perform a simple discrete convolution operation which for 2-D data can be represented by a matrix multiplication.
- *max pooling layers* that perform an input reduction by selecting one of many inputs, typically the one with maximal value

Implementation:

- KERAS library
- CNN — two convolutional layers with 32 filters and ReLUs, each, max pooling layer, fully connected layer of 128 ReLUs, and a fully connected output softmax layer.

Baseline Classification Acurracy

model	trainset	testset
MLP	1.00	0.98
CNN	1.00	0.99
RBF	0.96	0.96
SVM-rbf	0.99	0.98
SVM-poly2	1.00	0.98
SVM-poly4	0.99	0.98
SVM-sigmoid	0.87	0.88
SVM-linear	0.95	0.94
DT	1.00	0.87

Experimental Setup

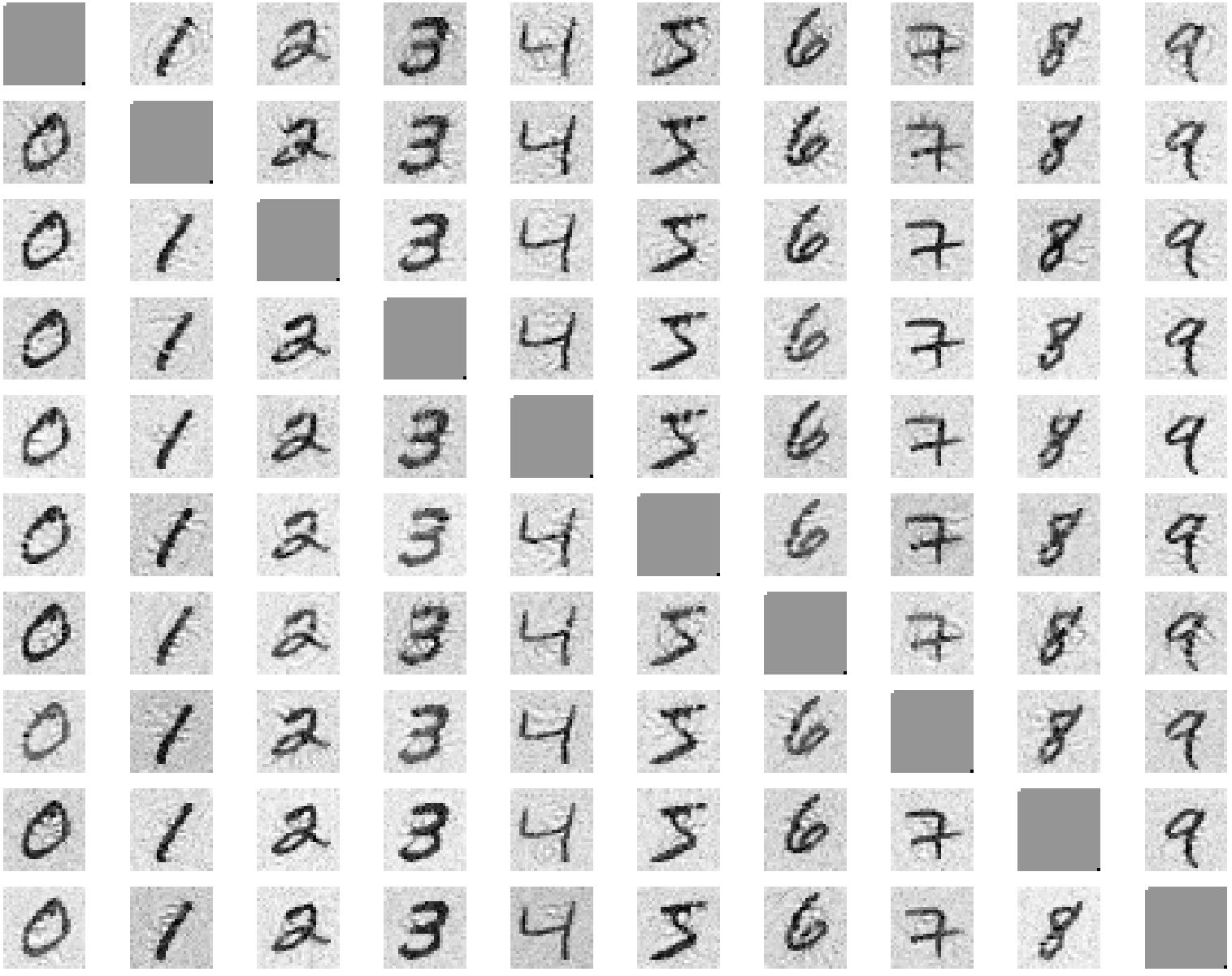
GA setup

- population of 50 individuals
- 10 000 generations
- crossover probability 0.6
- mutation probability 0.1
- DEAP framework

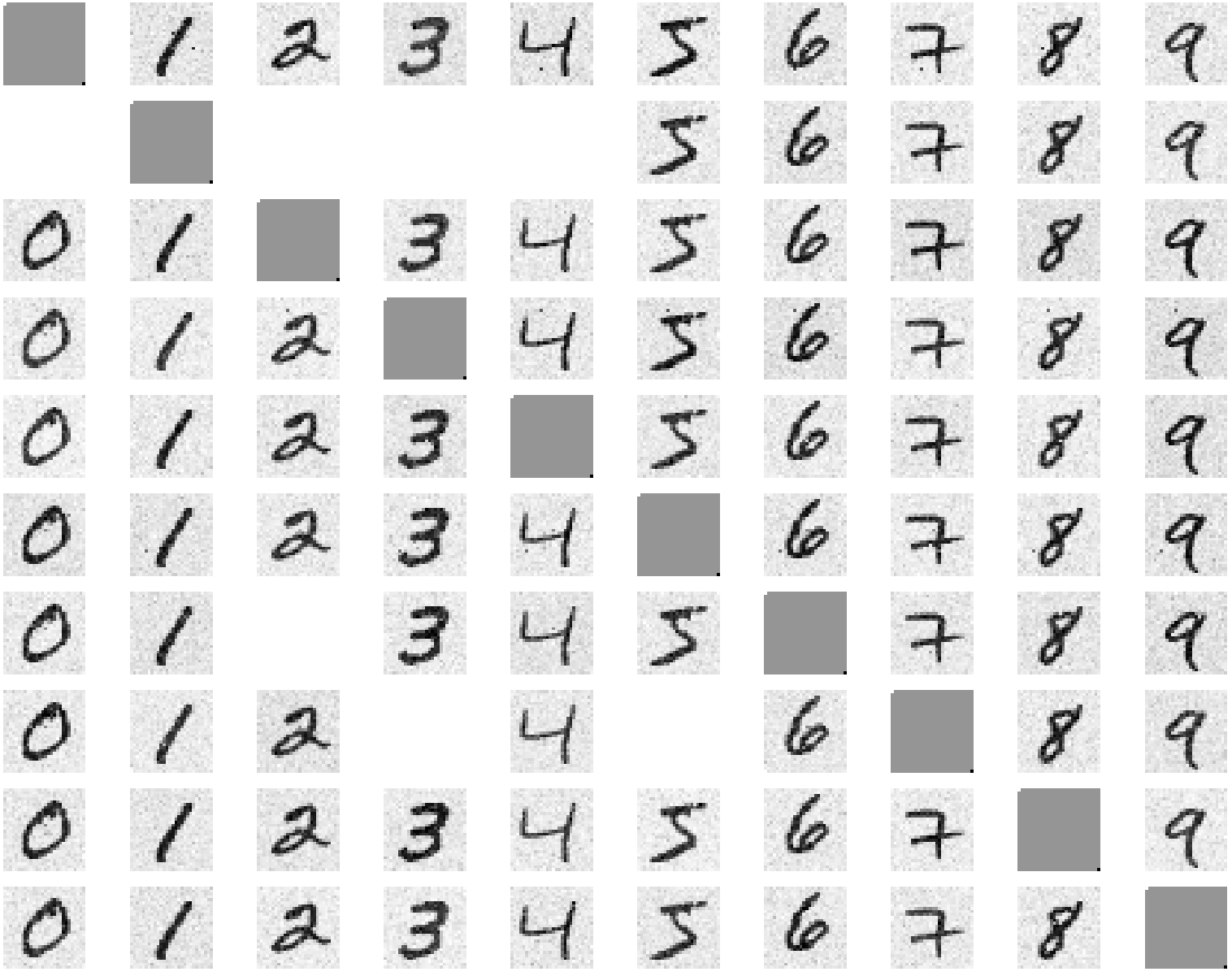
Images

- for 10 images from training set (one representant for each class)
- target: classify as zero, one, . . . , nine

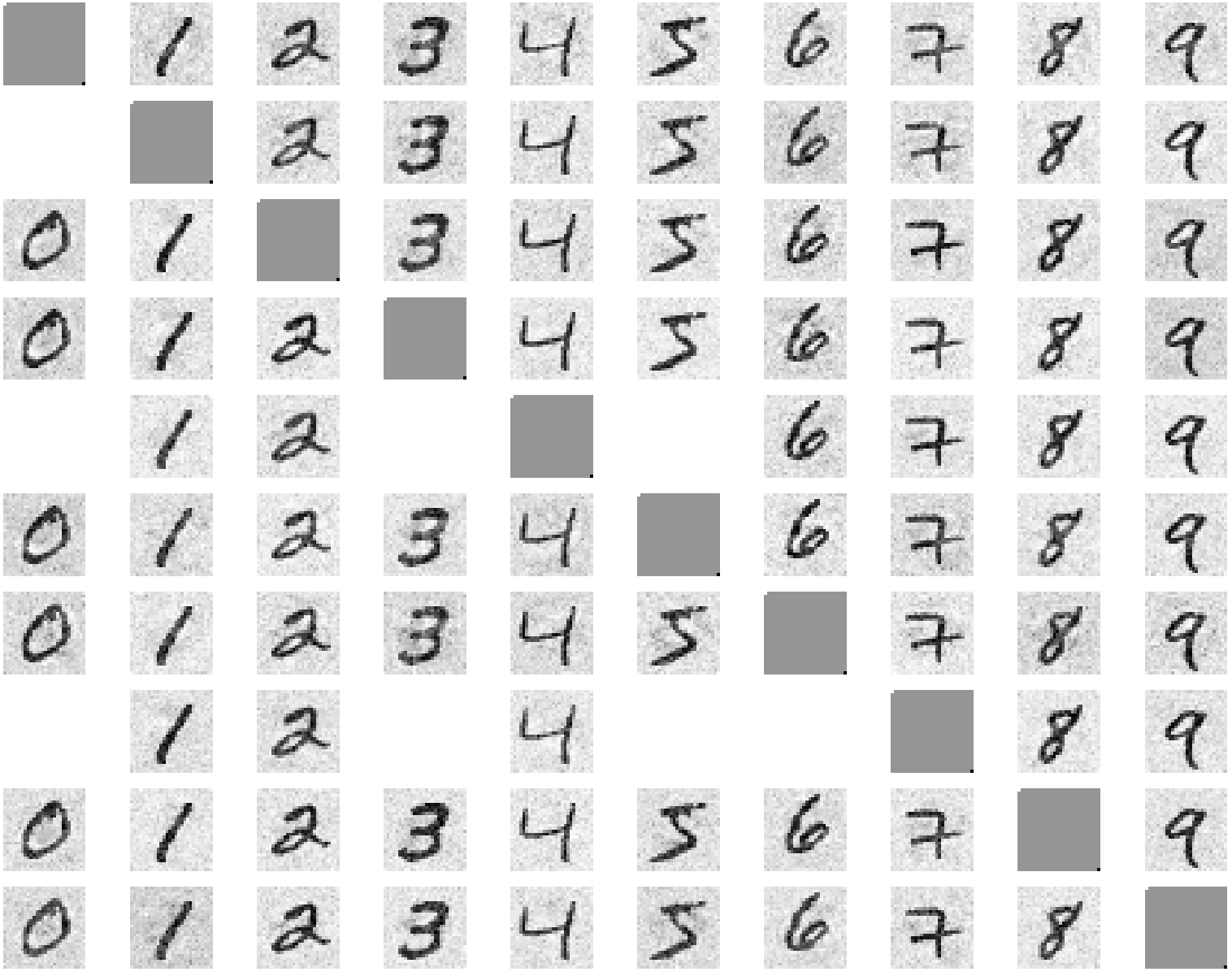
Evolved Adversarial Examples – CNN (90/90)



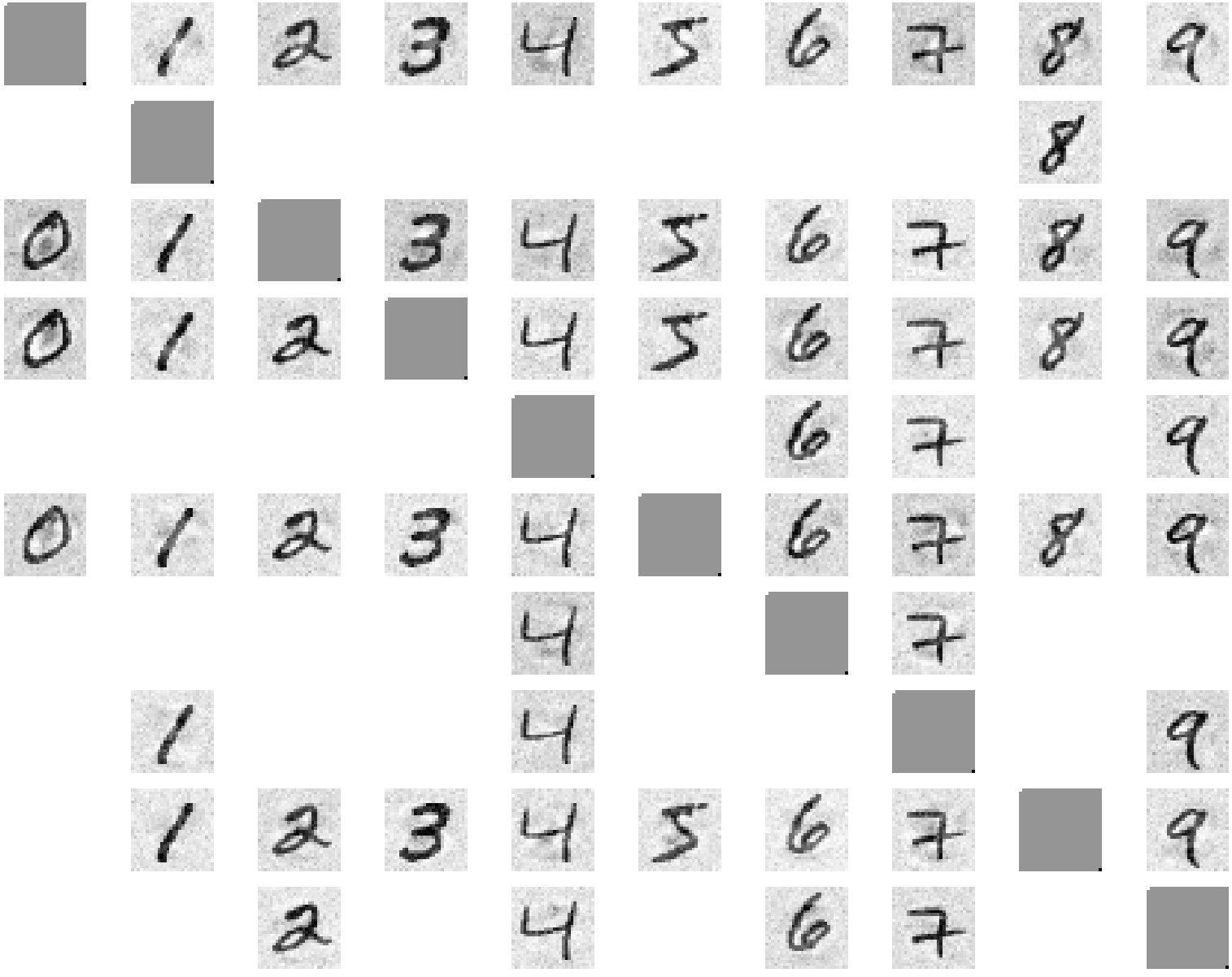
Evolved Adversarial Examples – DT (83/90)



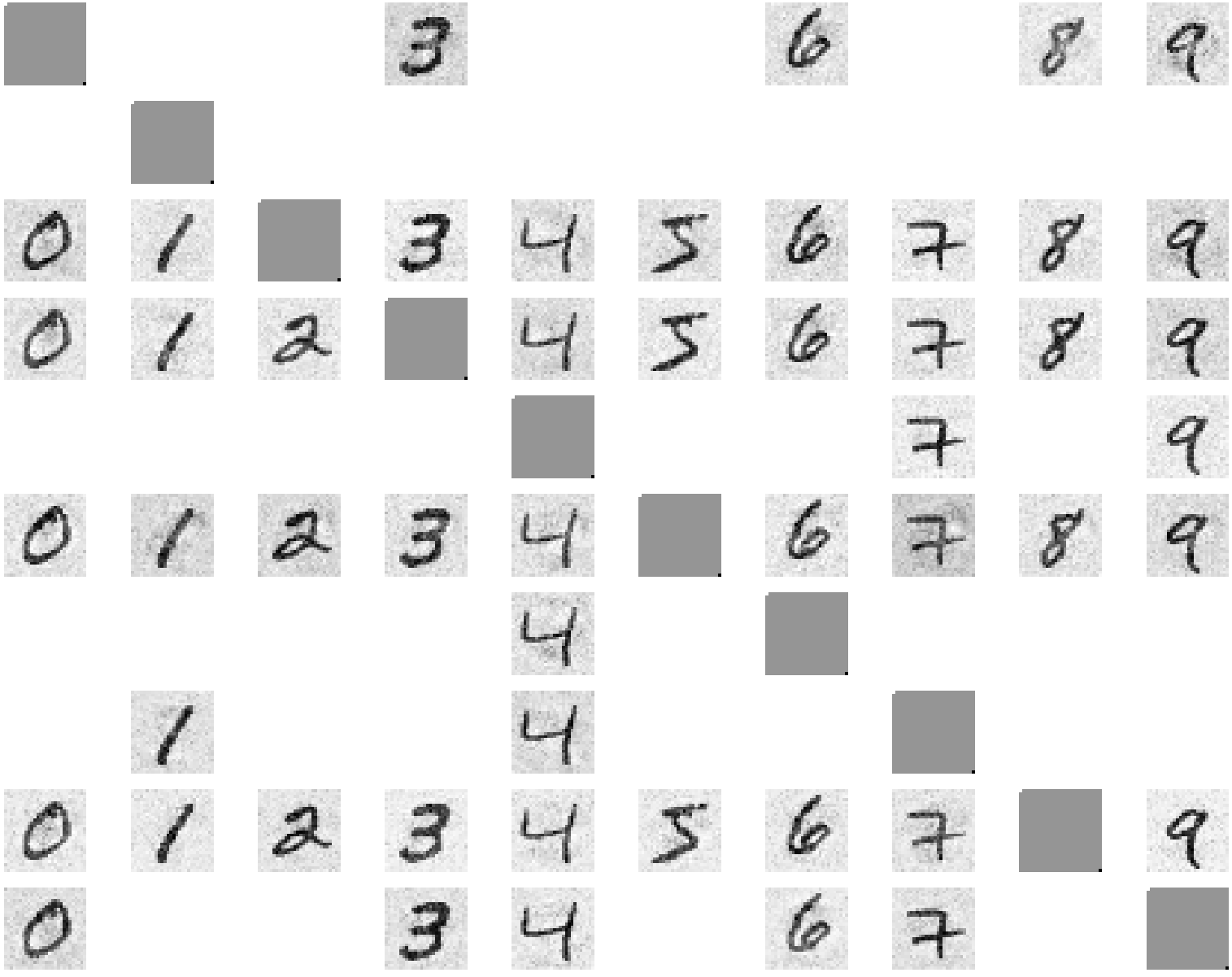
Evolved Adversarial Examples – MLP (82/90)



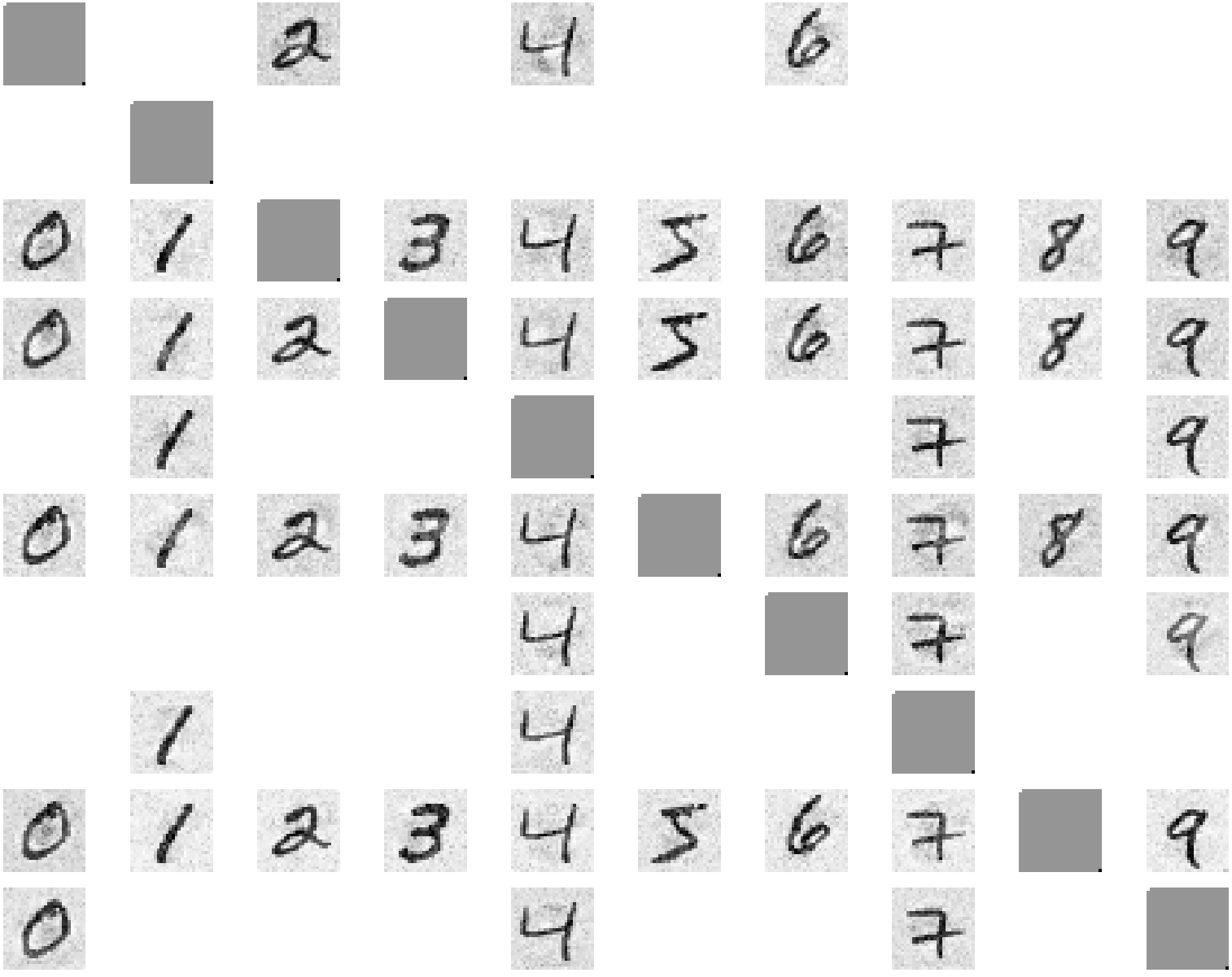
Evolved Adversarial Examples – SVM sigmoid (57/90)



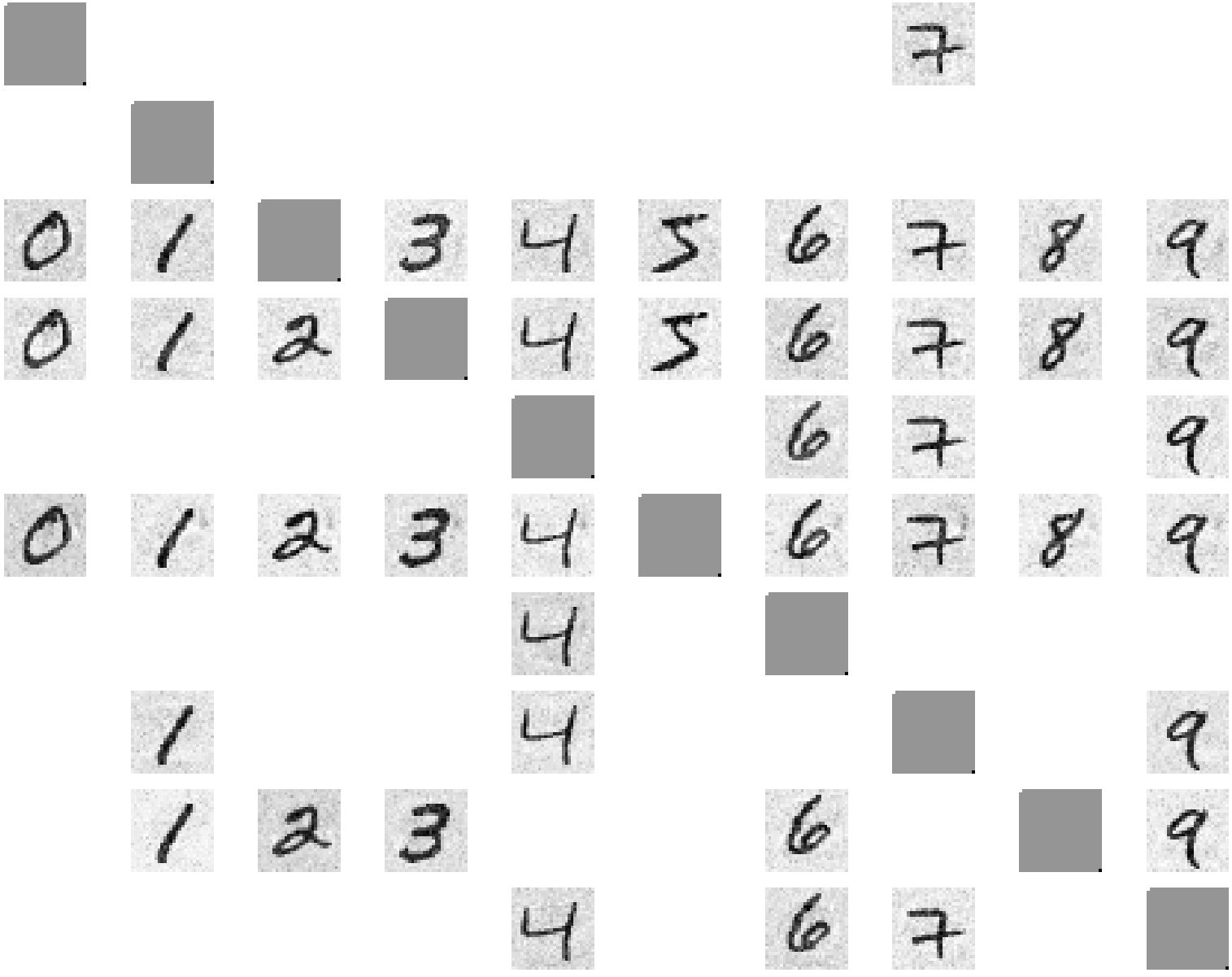
Evolved Adversarial Examples – SVM poly (50/90)



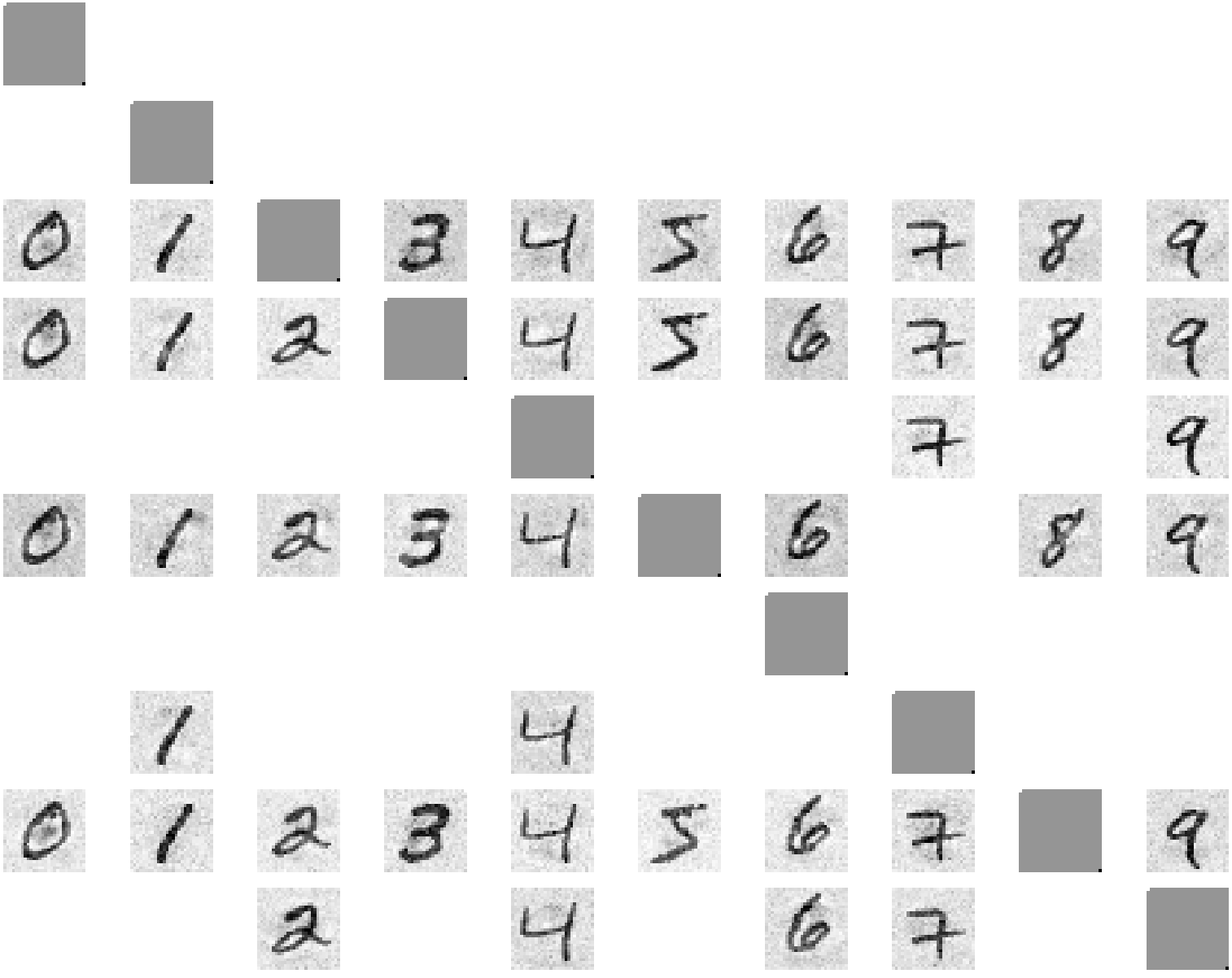
Evolved Adversarial Examples – SVM poly4 (50/90)



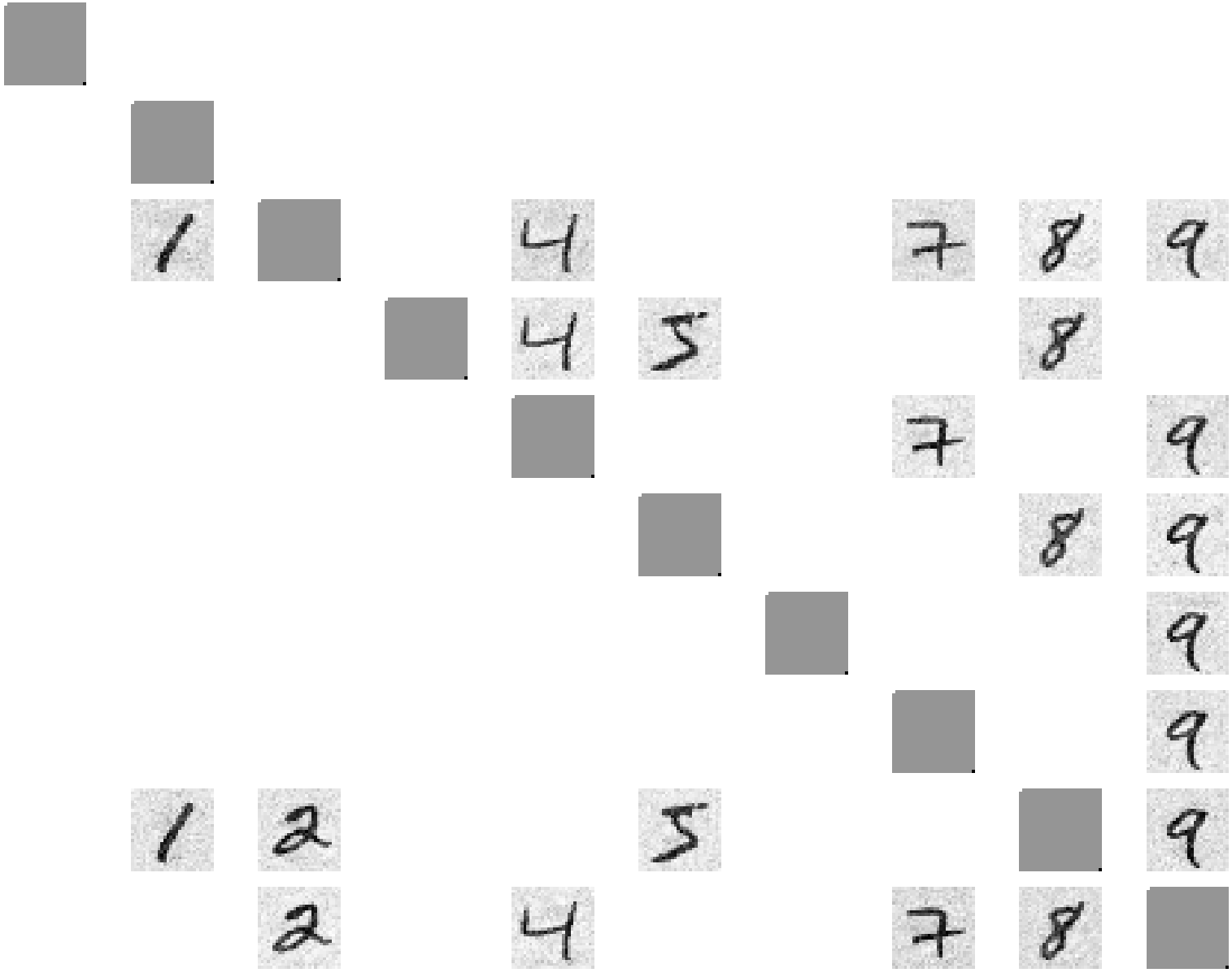
Evolved Adversarial Examples – SVM linear (43/90)



Evolved Adversarial Examples – SVM rbf (43/90)



Evolved Adversarial Examples – RBF (22/90)

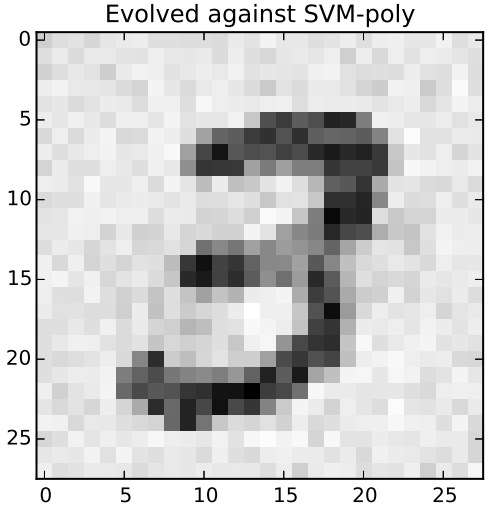


Experimental Results

- CNN, MLP, and DT were fooled in all or almost all cases
- RBF network was the most resistant model, but in 22 cases it was fooled too
- from SVMs the most vulnerable is SVM_sigmoid, most resistant is SVM_rbf and SVM_linear

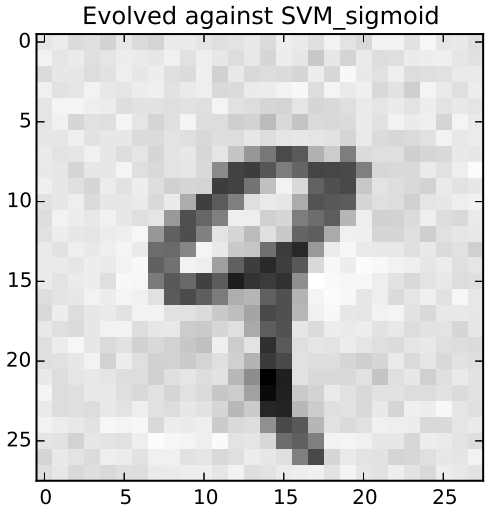
Generalization

- some adversarial examples generated for one model are also missclassified by other models



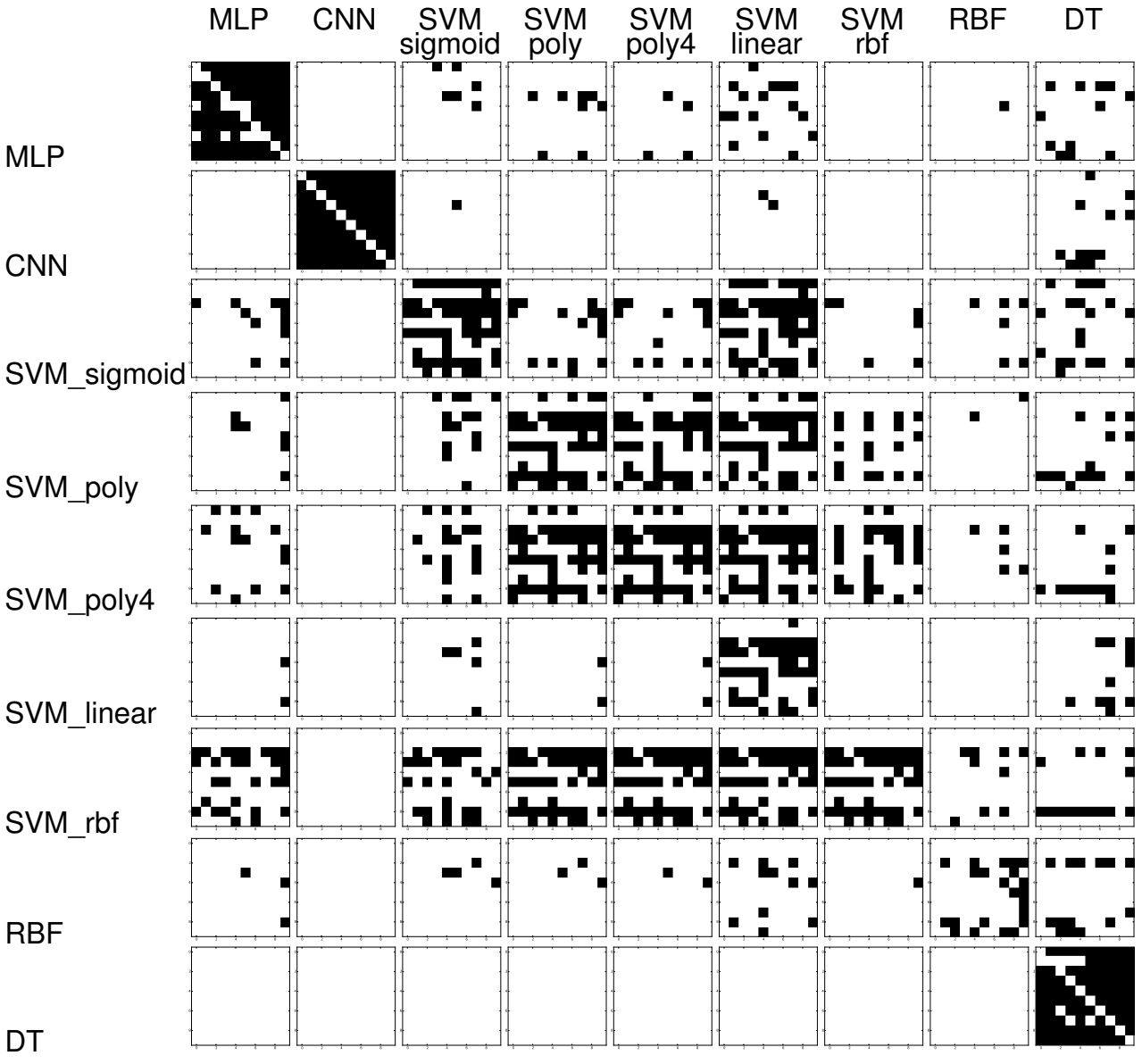
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
RBF	0.32	0.02	0.17	**0.86**	-0.01	-0.09	-0.09	-0.03	-0.12	0.01
MLP	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
CNN	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
ENS	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-rbf	0.00	0.00	0.00	**0.99**	0.00	0.00	0.00	0.00	0.00	0.00
SVM-poly	**0.87**	0.00	0.02	0.04	0.00	0.00	0.00	0.00	0.04	0.02
SVM-poly4	**0.38**	0.01	0.11	0.23	0.01	0.02	0.01	0.02	0.15	0.04
SVM-sigmoid	**0.55**	0.01	0.04	0.19	0.01	0.05	0.01	0.01	0.13	0.02
SVM-linear	**0.71**	0.01	0.02	0.06	0.01	0.02	0.01	0.01	0.15	0.01
DT	0.00	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00

Generalization



	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
CNN	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**
MLP	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00
SVM_sigmoid	0.00	0.01	0.00	0.00	0.01	0.01	0.00	0.00	**0.85**	0.11
SVM_rbf	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.98**	0.01
SVM_poly	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	**0.98**	0.02
SVM_poly4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**0.98**	0.01
SVM_linear	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	**1.00**	0.00
RBF	0.01	0.01	0.09	0.09	-0.10	0.06	0.07	-0.02	**0.44**	0.41
DT	0.00	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Generalization Summary



Adversarial vs. noisy data

- We tried to learn a classifier to distinguish between adversarial examples and examples that are only noisy.

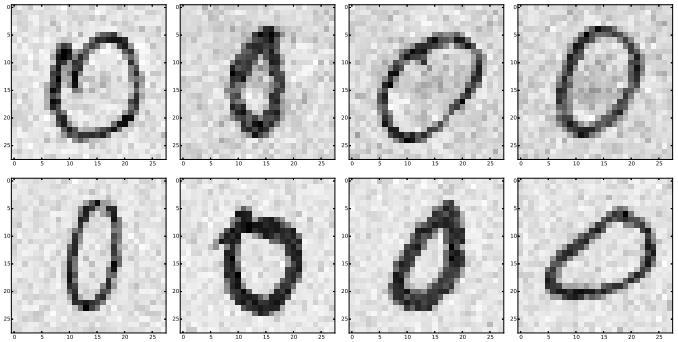


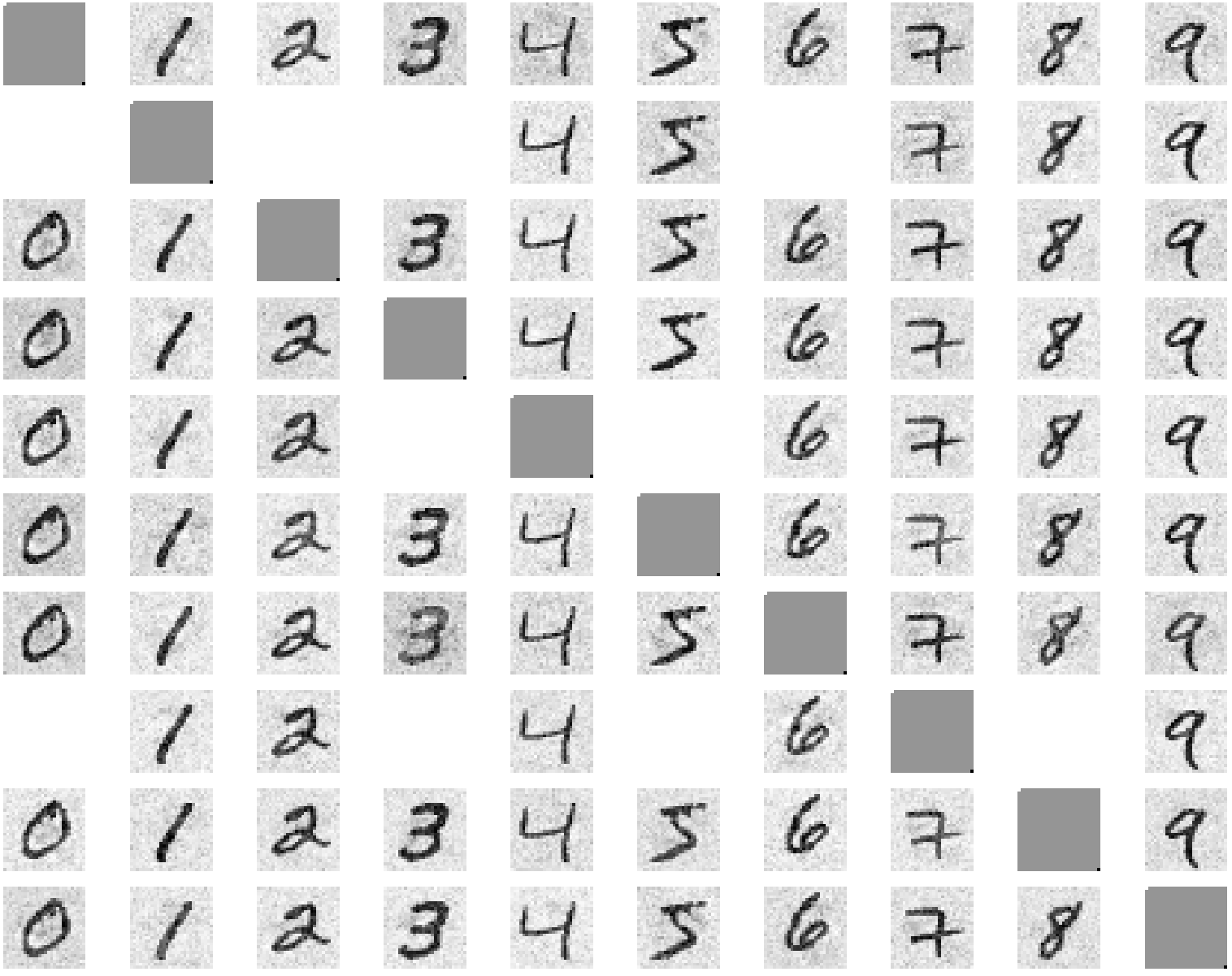
Figure : Digit zero —adversarials examples (top), noisy examples (bottom). Noisy examples were classified as zero by the MLP, adversarial examples as other class.

Adversarial vs. noisy data: results

- The data contains 22500 noisy examples and 19901 adversarial examples, and are randomly divided to training and test data (20% for test).

	precision	recall
SVM-rbf	0.888	0.843
MLP	0.923	0.912
CNN	0.964	0.925

New adversarial examples (for MLP)



Approaches robust to adversarial examples

- *Towards Deep Neural Network Architectures Robust To Adversarial Examples*.
  2015, Shixiang Gu, Luca Rigazio

- noise injection, Gaussian blur
- autoencoder
- deep contractive network

Gaussian blur of the input

- a recovery strategy based on additional corruption
- decrease error on adversarial data but not enough

Test error rates

	clean data			adversarial data		
blur kernel size	—	5	11	—	5	11
N100-100-10	1.8	2.6	11.3	99.9	43.5	62.8
N200-200-10	1.6	2.5	14.8	99.9	47.0	65.5
ConvNet	0.9	1.2	4.0	100	53.8	43.8

Autoencoder

- a three-hidden-layer autoencoder (784-256-128-256-784 neurons)
- trained to map adversarial examples back to the original data and original data back to itself
- autoencoders recover at leat 90% of adversarial errors

	N-100-100-10	N200-200-10	ConvNet
N-100-100-10	2.3%	2.4%	5.2%
N-200-200-10	2.3%	2.2%	5.4%
ConvNet	7.7%	7.6%	2.6%

- drawback: autoencoder and classifier can be stacked to form a new feed-forward network, new adversarial examples can be generated

Deep Contractive Network

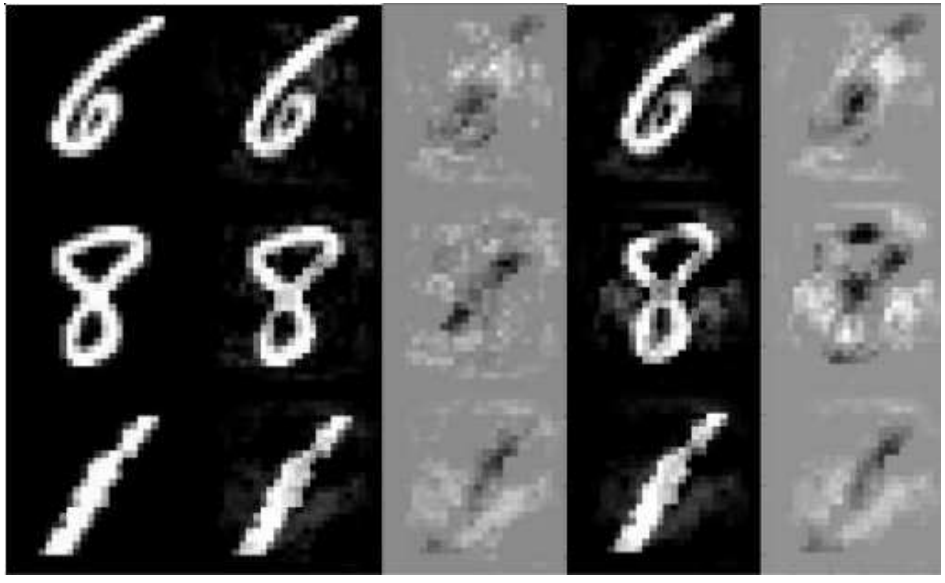
- layer-wise penalty approximately minimizing the network outputs variance with respect to perturbations in the inputs
- Deep Contractive Network (DNC) — generalization of the contractive autoencoder

$$J_{DNC}(\theta) = \sum_{i=1}^{m}(L(t^{(i)}, y^{(i)}) + \lambda||\frac{\partial y^{(i)}}{\partial x^{(i)}}||_2)$$

$$J_{DNC}(\theta) = \sum_{i=1}^{m}(L(t^{(i)}, y^{(i)}) + \sum_{j=1}^{H+1}\lambda_j||\frac{\partial h_j^{(i)}}{\partial h_{j-1}^{(i)}}||_2)$$

Deep Contractive Network – Experimental Results

	DCN		original	
model	error	adv. distortion	error	adv. distortion
N100-100-10	2.3%	0.107	1.8%	0.084
N200-200-10	2.0%	0.102	1.6%	0.087
ConvNet	1.2%	0.106	0.9%	0.095



Summary

- We have proposed a GA for generating adversarial examples for machine learning models by applying minimal changes to the existing patterns.
- Our experiment showed that many machine models suffer from vulnerability to adversarial examples.
- Models with local units (RBF networks and SVMs with RBF kernels) are quite resistant to such behaviour.
- The adversarial examples evolved for one model are usually quite general – often misclassified also by other models.

Thank you! Questions?