FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Štěpán Procházka

Adversarial Examples Generation for Deep Neural Networks

Department of Theoretical Computer Science and Mathematical Logic

Supervisor of the bachelor thesis:	Mgr. Roman Neruda, CSc.
Study programme:	Computer Science
Study branch:	General Computer Science

Prague 2018

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In .... date .... signature of the author

To whom it may concern, Thank You.

Title: Adversarial Examples Generation
for Deep Neural Networks

Author: Štěpán Procházka

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: TODO Abstract.

Keywords: machine learning adversarial examples evolutionary algorithms deep learning

Contents

Introduction

The effort to automate various processes in our lives has always been one of the key concepts of scientific research. The automation of mechanical work has already been solved to great extent by advances in engineering. On the contrary, automated processing of information has been solved just partially. Well defined tasks i.e., tasks with fully defined behaviour can be solved and the challenge lies just in effectivity of the solution. On the other hand, models solving tasks based on raw real word data, human level input and output or some degree of fuzziness are yet to be found or, if they exist, suffer from several shortcomings. One of those shortcomings is vulnerability to adversarial examples i.e., artificially created inputs misinterpreted by those models. In this thesis, we will cover the task of generating adversarial examples for the models used for classification in computer vision.

TODO 800 chars artificial intelligence -¿ machine learning -¿ sota deep learning

TODO 800 computer vision and classification tasks

1. Underlying Theory

In this chapter we will present theoretical background of various subjects relevant to this writing. We will give a compact overview of the task of image classification in computer vision with concrete examples of available datasets. We will cover various artificial intelligence paradigms with emphasis on deep learning for image classification and image generation, and evolutionary algorithms as a space search method. Adversarial example generation methods will be discussed in the following, separate chapter for their importance with respect to this work.

1.1 Classification Tasks in Computer Vision

The field of computer vision, sometimes percieved as a part of the artificial intelligence, examines the machine processing of imaging data. The basics of the field were laid down in the 1960s, the golden years of artificial intelligence, with ambitious goal to build systems with universal understanding of visual concepts. A multitude of subtasks of this ultimate goal were solved to a large extent; however the task as a whole remains to be solved at the time of writing. The computer vision is experiencing boom once again with the advent of deep learning and GPU accelerated computation.

Based on the previous approaches in CV, such as the use of convolutional filters, in conjunction with increasing computational power and advancements in machine learning, such as reinvention of gradient backpropagation, the convolutional neural networks emerged (see 1.2). Those models outperformed former state of the art methods i.e., SVMs with RBF kernels or SIFT with Fisher vectors, in terms of accurracy (Krizhevsky et al. [2012a]) and enabled wider range of tasks to be solved. The tasks that are being solved by CNNs in CV are image classification, object detection, semantic segmentation, image generation, style transfer, image captioning, etc. We will elaborate more on image classification tasks, available datasets and respective state of the art results.

1.1.1 Task Definition

The task of image classification lies in assigning one and only one label l from the set of possible output labels L to each input image I from the space of all possible images $\mathbf{I}$ of which we often think as a unit hypercube i.e., $[0,1]^{h \cdot w \cdot c}$ with h, w, c being input images height, width and number of channels (depth) respectively. Task with input varying in shape can be transformed to the canonical classification task by adding preprocessing step which unifies shape e.g., reshaping using bilinear interpolation or conversion to common colour space, or by building a multitude of solutions, each targeted at specific shape of input data.

1.1.2 MNIST like tasks

The whole family of datasets for benchmarking of image classification solutions has been created. For the sake of interchangeability all of them share the same input size of $28 \times 28 \times 1$ (rectangular grayscale image) and often same dataset

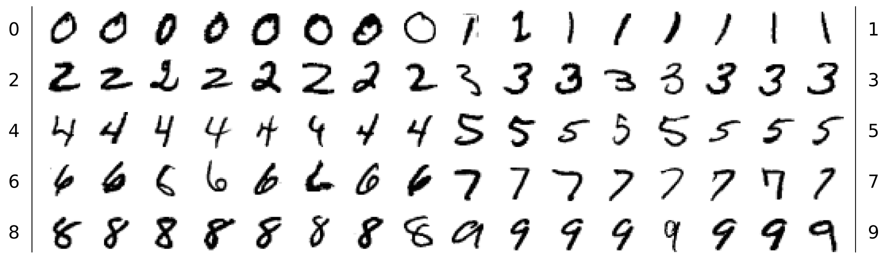

Figure 1.1: MNIST samples

size and structure with 60,000 training and 10,000 testing examples. Those datasets, particularly the original MNIST itself, became widely used benchmarking datasets for they are rather small in terms of the number of examples and their shape, enabling fast prototyping and experimentation.

MNIST The dataset comprises of handwritten digits – a selected subset of the National Institute of Standards and Technology [2010] database. The database consists of approximately 800,000 handwritten alphanumeric characters written by 3600 writers, namely high school students and the United States Census Bureau employees. The subset has been chosen so that the amount of samples written by students compared to the Census Bureau employees is equal in each dataset split. Moreover the sets of writers chosen for training samples and testing samples are disjoint. The chosen samples are normalized with following steps – each character is reshaped to fit $20 \times 20px$ patch preserving the aspect ratio and the center of mass of this patch is aligned with the center of larger $28 \times 28px$ patch creating final sample. The classification accuracy of over 99% was reached by Lecun et al. [1998] and their LeNet model. Due to increasing capabilities of machine learning approaches, MNIST became obsolete and is no longer a valid benchmarking dataset for performance comparison nowadays. 1.1

EMNIST The Extended MNIST (EMNIST) dataset contains alphanumeric characters and consequently is seen as an extension of the MNIST dataset. It comes from the same database and was created by mimicking the steps used for MNIST sample preprocessing. However, the procedure differs sligtly and consequently MNIST is not a subset of EMNIST. The dataset comes in several layouts. The EMNIST Complete consists of approximately $700,000$ training samples and $115,000$ testing samples each classified to one of 62 classes (10 digits, 26 lower-case and 26 upper-case letters). The EMNIST Merge is a variation of the complete EMNIST with visually ambigous letter classes e.g., *C, K, P, S* merged, leading to 47 classes in total. The EMNIST Balanced comprises of $112,800$ training and $18,800$ testing samples in 47 classes (using merging), with equal sample frequency across classes. Appart from those datasets, there are EMNIST Digits, EMNIST Letters and EMNIST MNIST, dataset sharing the layout of original MNIST dataset (Cohen et al. [2017]). Despite being substantialy harder task than MNIST, EMNIST is not widely used.

Fashion MNIST The Fashion MNIST shares the layout of MNIST dataset, having the same number of same shaped train and test examples as well as the

Figure 1.2: Fashion MNIST samples

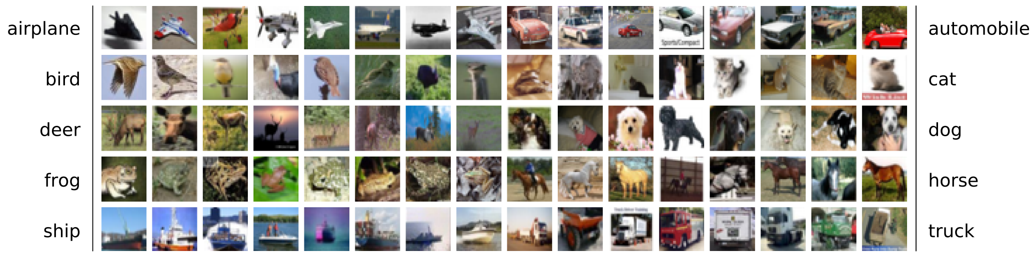

Figure 1.3: CIFAR 10 samples

same number of classes. The dataset comprises of grayscale thumbnails of pieces of clothes in ten classes *T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, Ankle boot* 1.2. This dataset seems to be promising replacement of the original MNIST as it is defining harder and more computer vision relevant task (images of real world objects rather than manmade symbols), while preserving moderate hardware requirements and model complexity demands. Xiao et al. [2017a]

1.1.3 Photo datasets

Due to the need to classify real world imagery i.e., data with substantial amount of noise, lacking quality or having non-trivial composition such as multiple objects in the scene, varying viewing angles or conditions, or heterogenous background), the datasets of real world photographs were created. Those datasets range from collections of thousands of low resolution thumbnails in dozens of classes to millions of reasonably large pictures in hierarchical systems of classes.

CIFAR The CIFAR dataset comes in two versions, namely CIFAR-10 and CIFAR-100, named after number of classes present. Both datasets share the same example shape i.e., 32×32 (rectangular color image) and dataset layout comprising of 50,000 training and 10,000 testing images with balanced class occurence. CIFAR-10 consists of following classes – *airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck* 1.3, while CIFAR-100 introduces hierarchical system of 20 superclasses e.g., *fish, small mammals, tress, large carnivores* each subdivided into 5 classes e.g., *fish – aquarium fish, flatfish, ray, shark, trout* (see Krizhevsky [2009].

ImageNet ImageNet is large image database, providing labeled data for image classification and object detection. The images are labeled according to WordNet® hierarchy (see Miller [1995]) and the whole database aim to provide approximately 1000 images for each class. ImageNet serves as a benchmark for current state of the art image classification architectures, which became popular being winning submisions to ILSVRC challenge (challange based on ImageNet dataset) – consisting of 1.2 million images in 1000 classes. Current state of the art results achieved by NASNet reach 3.8% top-5 error and 17.3% top-1 error.

1.2 Deep Learning in Image Classification

The deep learning is a branch of machine learning characterized by utilization of multilayered neural networks. Due to the complexity of deep networks and solutions tailored to solve tasks with fuzzy real world data, deep learning approaches are often not backed up by strong methematical theory; however they are proving successful in solving those tasks and provide current state of the art solutions in the field of computer vision, speech recognition and synthesis, machine translation, etc. Nowadays the goal of deep learning research lies in explanation of inner workings of deep models, which may result in constructing more complex and more capable machine learning solutions. See Goodfellow et al. [2016] for in depth introduction on deep learning.

1.2.1 Machine Learning

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. Mitchell [1997]

Machine learning is an Artificial Intelligence paradigm based on the concept of knoledge extraction from observed states or experience, followed by application of obtained knowledge to new observations. More formally the machine learning solution, often reffered to as *model* M, can be be percieved as the approximation of probability distibution P, optionaly conditional, of random variable $\mathtt{x}$, representing the task being solved. The model is often implemented by a parametric differentiable composite function M_θ and the process of finding the solution for a given task is called *training* and lies in finding the optimal set of parameters θ. The process of training relies on training data i.e., set of observed states, samples of random variable $\mathtt{x}$. The optimal set of parameters is searched for using maximum likelihood estimate principle i.e., the estimate of likelihood that the probability distribution of model P_{M_θ} matches the probability distribution of random variable $\mathtt{x}$. This process is implemented as a minimization of loss function, often using gradient descent algorithm. Given the characteristics of the training data, machine learning tasks can be split into the following two categories.

supervised training data takes form of (x, y) i.e., input – output or querry – answer example pairs. The goal is to approximate the conditional probability distribution $P(y|x)$ (*classification, encoding*); in case of *reinforcement* *learning*, the dataset itself is not available, and the ground truth comes in form of full description of the dynamic environment in which the model operates

unsupervised training data takes form of singletons x i.e., samples of unknown distribution and the the goal is usually to find unknown structure (*clustering*), be able to *generate* new samples i.e., approximate the probability distribution $P(x)$, or transform the data to some latent space with usefull properties (*compression, encoding, feature extraction*)

1.2.2 Loss Functions

Given the probability distribution $\bar{P}$ of the training data – samples of unknown distribution P, and probability distribution $\bar{P}_{M_\theta}$, the loss function quantifies the likelihood of $\bar{P}_{M_\theta}$ matching the distribution P, using the estimation $\bar{P}$. Frequently used loss functions are Kullback-Leibler divergence, often refered to as cross-entropy, used for the purpose of classification tasks; mean-squared error for regression tasks, or more advanced loss functions such as conditional random fields (*CRF*) loss or connectionist temporal classification (*CTC*) loss. Only the two former will be discussed in detail for their relevance to this writing.

Kullback-Leibler divergence For the discrete probability distributions, the *KL-divergence* has the following form.

$$D_{KL}(\bar{P} \parallel \bar{P}_{M_\theta}) = -\sum_i \log \bar{P}(i) \frac{\bar{P}_{M_\theta}}{\bar{P}(i)} \tag{1.1}$$

It holds that *KL-divergence* is always non-negative and equal to zero if and only if $\bar{P} = \bar{P}_{M_\theta}$. It is worth to be noted that *KL-divergence* is not symmetric, hence it is not a distance measure.

Mean squared error

$$MSE(\bar{P}, \bar{P}_{M_\theta}) = \frac{1}{n} \sum_{i=1}^{n} (\bar{P}(i) - \bar{P}_{M_\theta})^2 \tag{1.2}$$

It holds that *MSE* is always non-negative and equal to zero if and only if $\bar{P} = \bar{P}_{M_\theta}$.

1.2.3 Optimizers

The optimizer controls the process of finding optimum of the loss function. Unlike classical methods of mathematical optimization working with fully defined functions (or probability distributions they generate), optimizers in machine learning work with finite set of samples of unknown distribution. As a result, finding the global optimum on the training set is not the preferable goal, as it brings the issue of overfitting – relying on characteristics specific to the training data $\bar{P}$ which are not representative in context of the whole unknown distribution P. To fight overfitting, methods of *regularization* are often employed. Due to the hardware limitations, it is often not possible for the optimizer to work with the whole training set. As a result, sampling of dataset is employed in the form of batching, and the process of training is performed in *epochs* – iterations over the whole training set in batches. Optimizers frequently used during training of deep learning models are stochastic gradient descent (optionaly with momentum or Nesterov momentum), RMSProp, AdaGrad, Adam, etc.

Gradient Descent The gradient descent is a space search method. It navigates the space of the parameters of the machine learning model by iteratively taking steps in the opposite direction of the gradient of the loss function, hence minimizing it. The step size is driven by *learning rate* parameter. The algorithm does not guarantee finding global optimum. When using batching, we talk about stochastic gradient descent for we have access only to the approximation of the loss function in each step.

Algorithm 1 Stochastic gradient descent (optionaly with Nesterov momentum)

$\alpha \leftarrow$ learning rate
$[\beta \leftarrow momentum, v \leftarrow 0]$
repeat
  Sample a minibatch of m training examples $(x^{(i)}, y^{(i)})$
  $[\theta \leftarrow \theta + \beta v]$
  $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(M_theta(x^{(i)}), y^{(i)})$
  $[v \leftarrow \beta v - \alpha g]$
  $\theta \leftarrow \theta - \alpha g$
until early stopping criterion is met

Commands in brackets implements usage of Nesterov momentum with SGD. Each step, we first change parameters according to the accumulated momentum, than we estimate gradient of the loss function as in basic SGD, update momentum and perform step according to estimated gradient.

Adam The *Adam* optimizer is one of optimizers derived from basic SGD Kingma and Ba [2014]. It addresses the shortcomings of the stochastic gradient descent, mainly high variance in loss function gradient estimation and inability to effectively navigate certain landscapes. Adam keeps track of the first and the second moment estimates of the loss function gradient, using exponential moving average.

Adam estimates the mean of gradient in the same way as SGD with momentum. Moreover, the second moment estimate provides adaptive learning rate for each parameter in θ resulting in faster convergence during training and more stable behaviour.

1.2.4 Regularization

As stated before, the issue of overfitting is adressed by means of regularizations – methods, that improve model generalization ability (performance on unseen data), while not neccessarily changing the performance on training data. Frequently used algorithms are:

Algorithm 2 Adam optimizer

$\alpha \leftarrow$ learning rate
$\beta_1 \leftarrow$ mean decay rate, $\beta_2 \leftarrow$ variance decay rate
$s \leftarrow 0, r \leftarrow 0, t \leftarrow 0$
repeat
  $g \leftarrow \frac{1}{m}\nabla_\theta \sum_i L(M_theta(x^{(i)}), y^{(i)})$
  $t \leftarrow t + 1$
  $s \leftarrow \beta_1 s + (1 - \beta_1)g$
  $r \leftarrow \beta_2 r + (1 - \beta_2)g^2$
  $\hat{s} \leftarrow s/(1 - \beta_1^t)$
  $\hat{r} \leftarrow r/(1 - \beta_2^t)$
  $\theta \leftarrow \theta - \frac{\alpha}{\sqrt{\hat{r}+\epsilon}}\hat{s}$
until early stopping criterion is met

early stopping is policy terminating training process if model performance on validation set visibly stalls or worsens

p-norm regularization adds additional term in loss function computing p-norm of model parameters, forcing them to maintain mean close to zero and low variance; $l2$ and $l1$ norms are often used

dataset augmentation stands for careful application of transformations to training data providing mean of enlarging training set, and consequently improving model performance; shifting, flipping, random cropping, addition of random noise are broadly used augmentation methods, they need to be tailored to specific task to preserve label of augmented example

ensembling lies in training multiple models with different initialization, different training process or even of different architectures and employing a voting scheme for those models often leads to better results; however this approach comes with significant computational power demands

dropout masks internal featuremaps (hidden state) randomly during training, forcing the layers to perform well even with noisy inputs, pushing the decision boundary between classes far from presented examples

1.2.5 Layers

Neural networks are often thought of as layered structures with layers being transormations of tensors. The deep neural networks are composed of tenths or even hundreds of those layers. Various types of layers serve for feature extraction, non-linear transformation, normalization, reshaping, etc.

Dense The most basic and arguably the first layer invented is a dense layer, computing biased linear combination of inputs. Given the input vector x_{in} of lenght s_{in}, parameters W as weight matrix of shape $s_{out} \times s_{in}$ and b bias vector of length s_{out}, the output of dense layer of size s_{out} corresponds to 1.3.

$$L_{W,b}(x_{in}) = Wx_{in} + b \tag{1.3}$$

Activation functions Linear combinations itself are not able to form strong enough models, Cybenko [1989] proved, that added non-linearity, in their case *sigmoid* 1.4 function, theoretically enables construction of universal approximators. This theorem, often referred to as the universal approximation theorem, has been later proven for other non-linear functions e.g., *tanh*, rectified linear unit $ReLU$ 1.5 and its modifications, etc. (see Sonoda and Murata [2015]). Those are widely used in machine learning as an activation functions.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.4}$$

$$f(x) = \begin{cases} x, & \text{for } x > 0 \\ 0, & \text{for } x \leq 0 \end{cases} \tag{1.5}$$

Activation function as a neural net layer is then straightworward elementwise application of it.

Convolution A multitude of machine learning tasks process data e.g., images, natural language samples, audio, with inherent spatial properties – mainly context locality. Convolutional layers use fixed size sliding window (*kernel*) to perform discrete convolution, in order to globally extract local context. Given the input tensor X_{in} of shape (w, h, d_{in}), the 2-dimensional convolution of output depth d_{out}, with kernel tensor K of shape $(w_K, h_K, d_{out}, d_{in})$ and bias vector b of length d_{out} corresponds to 1.6.

$$Conv2D(X_{in})_{x,y} = \sum_{i=1}^{w_K} \sum_{j=1}^{h_K} W_{i,j} X_{in_{x+i-\lceil w_K/2 \rceil, y+j-\lceil h_K/2 \rceil}} + b \tag{1.6}$$

n-dimensional convolution for different n analogicaly. As the convolution crops the input tensor proportionally to size of the kernel, padding of input tensor may be used as a preprocessing step, to maintain constant featuremap shape. To perform featuremap downsampling, we can compute convolution only in evenly spaced positions, the distance between those positions is referred to as stride.

Pooling Another way of downsampling is pooling. Unlike strided convolution, pooling is often parameterless and instead of performing convolution with trainable kernel, it applies given function to sliding window (channel-wise). Frequently used functions are arithmetical *average* and *maximum*.

Normalization This type of layer fights internal covariate shift, undesirable effect of model hidden representations not having stable probability distribution, by centering and scaling featuremaps to have zero mean and unit variance. Usage of normalization often results in faster and more stable training as trainable layers need not compensate for, possibly significant, changes in properties of outputs of preceding layers. Normalitation layer keeps track of first and second moment estimate often using exponential moving average. There is a multitude of normalization layers varying in a way those moments are computed. Normalization yields moderate regularization effect.

batch normalization takes place over all but last axis (channel) of featuremap; bigger batches benefit more (see Ioffe and Szegedy [2015])

layer normalization takes place over all but first axis (example) of featuremap; suitable even for smaller batches

group noramlization takes place over group of channels per example, multiple groups are formed, forcing features of similar moments, ideally features with similar semantics, to share the same group (see Wu and He [2018])

1.2.6 Known Architectures

The advent of deep learning and its soaring popularity has been boosted by outstanding results achived with deep learning models, beating then state of the art methods by a large margin. Arguably *AlexNet* was the first architecture to show the potential of convolutional neural networks, followed by gradually deeper architectures *VGGNet, ResNet and DenseNet*. Apart from image classification, deep learning models dominated image segmentation, natural language processing tasks e.g., machine translation, speech recognition, synthesis etc. Those architectures are out of scope of this writing.

AlexNet Built from convolutions of size $11 \times 11, 5 \times 5 and 3 \times 3$, the 8-layered *AlexNet* with 62.3 million parameters scored 15.3% top-5 accuracy in *ILSVRC 2012* competition. It uses *max* pooling for downsampling, $ReLU$ activation, dropout for regularization after dense layers, and SGD with momentum as an optimizer. See Krizhevsky et al. [2012b] for more details.

VGGNet With twice as many layers as *AlexNet* and more then double parameter count of 138 millions, *VGGNet* scored top-5 accuracy of 7.3%. It uses 3×3 convolutions exclusively, $ReLU$ activation, dropout for regularization after dense layers, and SGD with momentum. See Simonyan and Zisserman [2014] for more details.

ResNet Trying to train even deeper models, the issue of vanishing gradients arose. He et al. [2015] came up with residual shortcuts, connections bypassing convolutions, which enabled gradient flow. With that improvement, *ResNet-152* with 152 layers scored top-5 accuracy of 3.6% with 60.2 million trainable parameters. It uses 3×3 convolutions, $ReLU$ activation, batch normalization between each convolution and activation, dropout after dense layers, and SGD with momentum.

DenseNet Further exploiting the performance gain of residual shortcuts usage, Huang et al. [2016] created *DenseNet* architecture. Residual connections do not bypass just one layer at a time, but connects every preceding layer to the current one, forming a so called *dense block*, those blocks are interconnected with transition layers, which perform downscaling of featuremaps. Several improvements as bottlenecking and feature compression using 1×1 convolutions are introduced. It uses 3×3 convolutions, $ReLU$ activation, dropout and batch normalization between each convolution and activation and SGD with Nesterov momentum for training.

1.3 Evolutionary Algorithms as Space Search Algorithm

Evolutionary algorithms represent one of heuristic search algorithms and are inspired by the process of natural evolution. Following the work of Charles Darwin, we can think of evolution as of the process of gradual improvement of individuals of some population, with respect to a meassure of fitness, given by environment the population lives in. The key principle is, according to Darwin, the natural selection i.e., the survival of the fittest, which together with process of reproduction and heredity leads to propagation and spread of promising traits in succeeding generations. Formalising those principles, we obtain heuristic search algorithm by simulating natural evolution. For in depth introduction see Engelbrecht [2007].

Assume a task with objective O on space S, then:

population is a subset of space to be searched

$$P \subset S$$

individual is an element of population

$$i \in P$$

fitness is a function of real valued meassure of fitness depending on individual, often function of the objective function of task to be solved

$$F : i \mapsto \mathbb{R}, F : O(i) \mapsto \mathbb{R}$$

operator is a function converting one or more populations to new one, often in element-wise (per individual) manner

$$Op : P_{in_1}, P_{in_2}, \ldots, P_{in_n} \mapsto P_{out}$$

epoch is one step of evolutionary algorithm consisting of application of sequence of operators to current population, producing new population

$$P_{t+1} = Op_n(Op_{n-1}(\ldots Op_1(P_t)))$$

The evolutionary algorithm stops if the stopping criterion is met i.e., if some individual of current population, or the whole population itself, reach sufficient performance with respect to objective O.

1.3.1 Operators

The ability of evolutionary algorithms to effectively search given space lies in application of suitable operators. Those operators can be divided into three categories – *reproduction* operators e.g., *cross-over* internally combine several individuals into new one, *mutation* operators add random perturbations to individuals, with the idea of exploring genes not present in current population and finally *selection* operators mimicking the natural selection based on fitness of individuals. Following paragraphs elaborate on several most frequently used operators.

N-point Cross-over Assuming the population of individuals being vectors of genes $i \in G_1 \times G_2 \times \cdots \times G_l$, n-point crossover takes two distinct individuals i, j producing offspring u, v by randomly partitioning parent vectors into $n + 1$ segments, and swapping odd ones. Crossover with $n+1 = l$ is called the universal crossover and instead of partitioning the genome to segments and swapping them based on parity, it decides for each gene, whether to swap it or not randomly. For individuals being multidimensional matrices, the partitioning can be defined analogicaly, by partitioning each axis separately. Various modifications of cross-over are used based on the type of genes. For real valued ones, averaging (optionaly weighted) instead of swapping is sometimes used.

Random Mutation Assuming the population of individuals being vectors of genes $i \in G_1 \times G_2 \times \cdots \times G_l$, each gene g_k of individual i is mutated, with probability p_{gene} by drawing a sample from given distribution – this may be uniform distribution over gene domain G_k (unbiased mutation), normal distribution centered in g_k with given variance in case of real valued genes, etc.

Roulette Wheel Selection Roulette wheel selection represents one of stochastic selection operators. Given population P of individuals $i_1, i_2, \ldots, i_n$ we can construct an imaginary roulette wheel, where each individual is represented by a slot of size proportional to its fitness normalized by aggregate fitness of the whole population. Selected population is then sampled from this roulette wheel. As any individual may be chosed multiple times, small populations suffer from high variance in roulette wheel sampling, this issue is addressed by stochastic universal sampling algorithm.

2. Adversarial Examples for Deep Learning Models

Deep learning models are not perfect and certain amount of input data gets miss-classified naturally due to the insufficient accuracy of models; however Szegedy et al. [2013] showed that misclassification can be achieved artificially by adding small perturbations designed to fool the model to the previously correctly classified examples. Those inputs are called adversarial examples. We will focus on adversarial examples for image classification task. The task of generating adversarial example for input x, model M_θ, such that $M_\theta(x) = l$ is the correct label, lies in finding the perturbation η, such that $M_\theta(x + \eta) = M_\theta(x') = l'$, with l' being target class for the adversarial attack.

2.1 Taxonomy

We can characterise adversarial attacks by following, mutually independent, properties

- type of attack target

  targeted for l' is preset to given class

  non-targeted if it holds that$l' \neq l$

- number of original examples to be attacked by single η

  singleton attack for single x such that $M_\theta(x + \eta) = l'$

  multi attack for multiple $x \in X$ such that $\forall x \in X : M_\theta(x + \eta) = l'$

  universal attack for class l such that $\forall x : M_\theta(x) = l \implies M_\theta(x + \eta) = l'$

- the amount of knowledge of the targeted model

  white-box attack, where the adversary has full (read-only) access to model architecture and parameters

  black-box attack, where the adversary has access only to the classification outputs (predicted class, scores)

  surrogate attack, where the adversary treats target model as a black-box, yet has full access to substitute model i.e., model sharing some characteristics of target model e.g., task, training data, architecture, output probability distribution, etc.

- intensity of perturbation with respect to given measure – often $l1$ or $l2$ norm, or *PASS* (Rozsa et al. [2016]) and *SSIM* (Wang et al. [2004]) visual similarity measures

  constrained for perturbation fitting in given range with respect to some meassure

  optimized with perturbarion intensity being minimized

2.2 Gradient based methods

Thanks to inherent property of deep learning models – differentiability, gradient based methods are easy to be used and perform well in white-box and reasonably well in surrogate scenarios. More of those methods have been invented, namely *L-BFGS* using Broyden–Fletcher–Goldfarb–Shanno optimization algorithm, fast gradient sign method and its modifications, feature adversary, one-pixel attack, etc.

Fast Gradient Sign Method The fast gradient sign method computes gradients of loss function of the target model with respect to target example x and class l'. The sgn of gradients is taken and multiplied by ϵ, the step size and subtracted from x (targeted attack) – this way we move the target image in direction of rising target class prediction 2.1. In case of non-targeted attack, gradients are computed with respect to source class l and instead of subtraction we use adition to move the target image in direction of falling source class prediction.

$$x' = x - \epsilon sgn(\nabla_\theta L(M_\theta(x), l')) \tag{2.1}$$

The FGSM can be used iteratively with $x_t = x'_{t-1}$. Optional clipping after each step may take place in case of constrained attack. Versions of FGSM with momentum exists – generating the adversarial example the same way as the training of deep learning models optimizes the parameters.

2.3 Generative models

Appart from methods based on backpropagation, usage of deep generative models for adversarial example generation has been researched. Zhao et al. [2017] trained generative adversarial network on task dataset mapping random latent space to images – generator G and invertor mapping images to latent space of generator I – inverter. The adversarial examples are generated by finding z' close to $z = I(x)$, such that $M_\theta(G(z')) = l'$. This approach genererate adversarial examples more natural to human observer as we are perturbing the hidden representation and not the final output.

3. Our Solution

3.1 Pure Evolutionary Algorithms

3.2 Hybrid Methods

4. Experiments

In order to empirically assess the performance of proposed solutions we have carried out a multitude of experiments on the Fashion MNIST dataset. This particular dataset was chosen for it is simple enough to enable thorough examination of various scenarios using k-fold cross-validation and measurements of results on statistically significant number of examples (often whole Fashion MNIST test set), yet more complex then MNIST and the like (1.1.2). Following the results of Fashion MNIST benchmark (Xiao et al. [2017b]) we have chosen two high scoring model architectures of varying nature – namely:

SimpleNet Simple wide shallow network with three convolutional layers of depth 32, 64 and 128 respectively and one hidden dense layer of size 128. Each convolutional layer uses square kernel of size 3 and stride 1 with *same* padding and is followed by batch normalization and ReLU activation. Max pooling with kernel of size 2 and stride 2 is added in-between each pair of convolutions for subsampling. Flattened output of last convolutional layer is followed by ReLU activated hidden dense layer and final output layer of size corresponding to the number of classes. Dropout with rate 0.3 is added after each max pooling and in front of hidden and output dense layers. Model is trained with Adam optimizer (Kingma and Ba [2014]) with default (recommended) parameters for 128 epochs. Initial learning rate 0.001 is divided by 5 in 50% and 75% of training process. To the best of our knowledge the name *SimpleNet* does not refer to any well-known model architecture and was chosen for easier distinction of used architectures when refering to them. Whole model has approximately 900k trainable parameters.

DenseNet Specifically *DenseNet-BC* of depth 52 with growth rate $k = 8$ and 0.5 compression factor, representing deep narrow convolutional network. Dropout of rate 0.2 is used as proposed by authors as well as recommended training parameters i.e., SGD with Nesterov momentum of 0.9 and learning rate of 0.1 divided by 10 in 50% and 75% of training process which takes, again, 128 epochs. Whole model has approximately 120k trainable parameters. See Huang et al. [2016].

Both models use weight decay of 0.0001 as additional regularization method as well as standard input preprocessing - mean subtraction and standard deviation division with channel-wise precomputed moments on training data (TODO citation). Batch size of 128 is used during training and no early stopping method is employed.

Training of both architectures is carried out using stratified 10-fold cross-validation resulting in 20 target models trained with varying random initialization on differing training sets. From the test set accuracy boxplot 4.1, we can see that densenet outperforms simplenet by approximately 0.8%. The confusion matrices of median models are very similar for both architectures.

Experiments overview In the following sections we are going to examine following approaches towards generating adversarial examples for image classifiers.

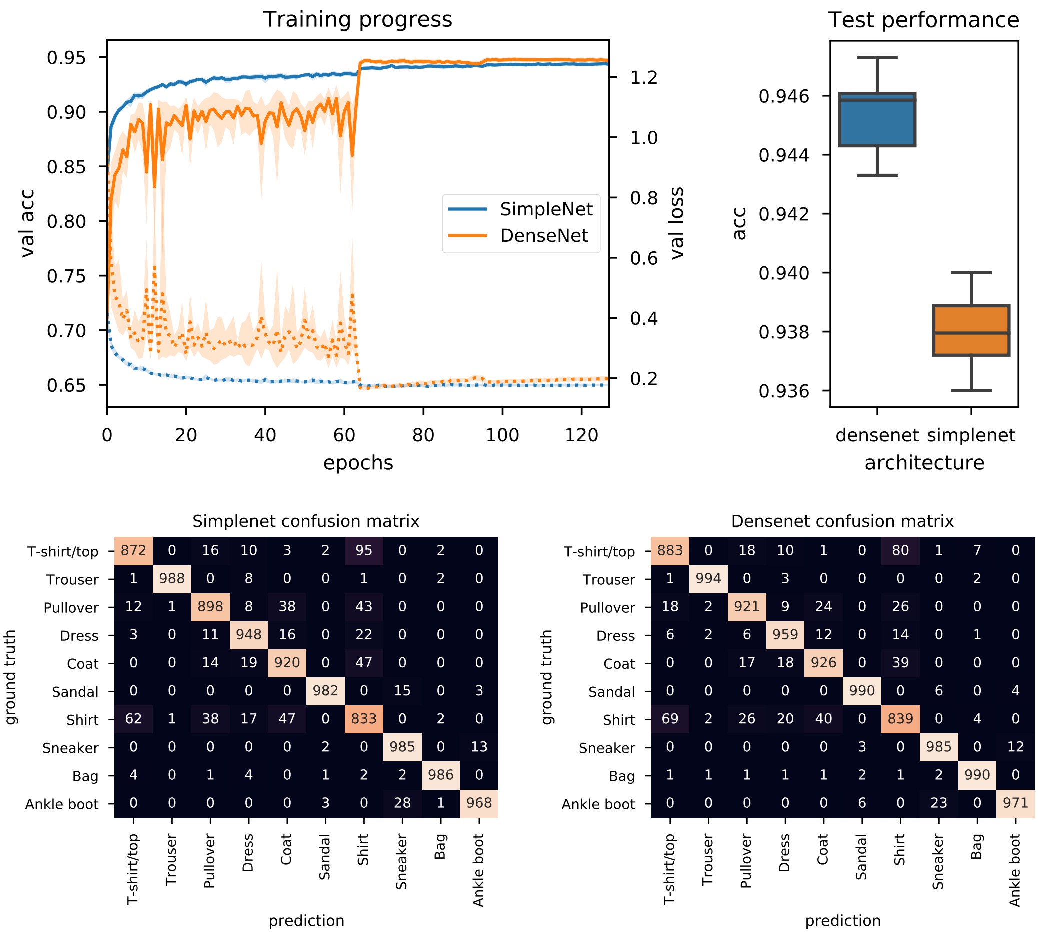

Figure 4.1: Cross validated training results

Firstly, white-box fast gradient sign method attack serving as both implementation sanity check and baseline is carried out. Next the transferability of adversarial examples is examined using surrogate models. Those approaches include transfering adversarial examples between models trained on the same dataset – the most naive approach, training surrogate model on outputs of target model and finally training focused surrogate models as a binary classifiers for specific classification category in one-vs-all manner. For each approach based on use of surrogate models, the transferability of adversarial examples between models of the same architecture as well as models of differing architectures is assessed. Appart from pure gradient methods, evolutionary algorithms are employed as a space search method, generating adversarial examples for target model. Finally using both gradient based methods and heuristic space search together, the performance of hybrid methods is measured.

For some experiments one *SimpleNet* and one *DenseNet* model is chosen, taking median models with respect to accuracy, we will refer to them as *median SimpleNet/DenseNet*. In the same manner a single class is chosen, median in terms of model prediction accuracy and attack performance, for use in some experiments – this proves to be the *dress* class for Fashion MNIST dataset.

Each approach is tested under combination of following conditions - either non-targeted and targeted attack to each class is carried out, while at the same time generating single or universal adversarial pattern is tested. Attack is constrained to a maximum of 0.1 pixel-wise difference between targeted and adversarial examples, which corresponds approximately to 26 shades of gray difference in either direction.

4.1 Fast Gradient Sign Method Approaches

Iterative FGSM is applied to each example in Fashion MNIST test set. FGSM with step size of $\frac{1}{255}$ is used, which corresponds to shift by one shade of grey in resulting adversarial example. The following experiments compare adversarial attack performance for various model architectures, target model knowledge available to adversary and character istics of the attack itself.

4.1.1 White-Box Attack

The white-box iterative FGSM is run against the target models directly. Following box plots show the distribution of the number of steps (*y-axis*) needed to be taken to obtain adversarial example from source label (*x-axis*) to target (*plot title*), those results are collected on the whole Fashion MNIST test set using cross validated target models i.e., for each class – architecture pair, the median step count for each test set example is taken, resulting in 1000 datapoints. Only valid datapoints i.e., those representing successfully generated adversarial examples are used to render each box. Presented heatmaps than show success rate of attack for each source – target label pair for given architecture (*plot title*). Blank cell in heatmap represents maximal possible value i.e., 1000.

Singleton Following from presented results 4.2 we can see that white box attack against single example proves to be successful in both targeted and non-targeted

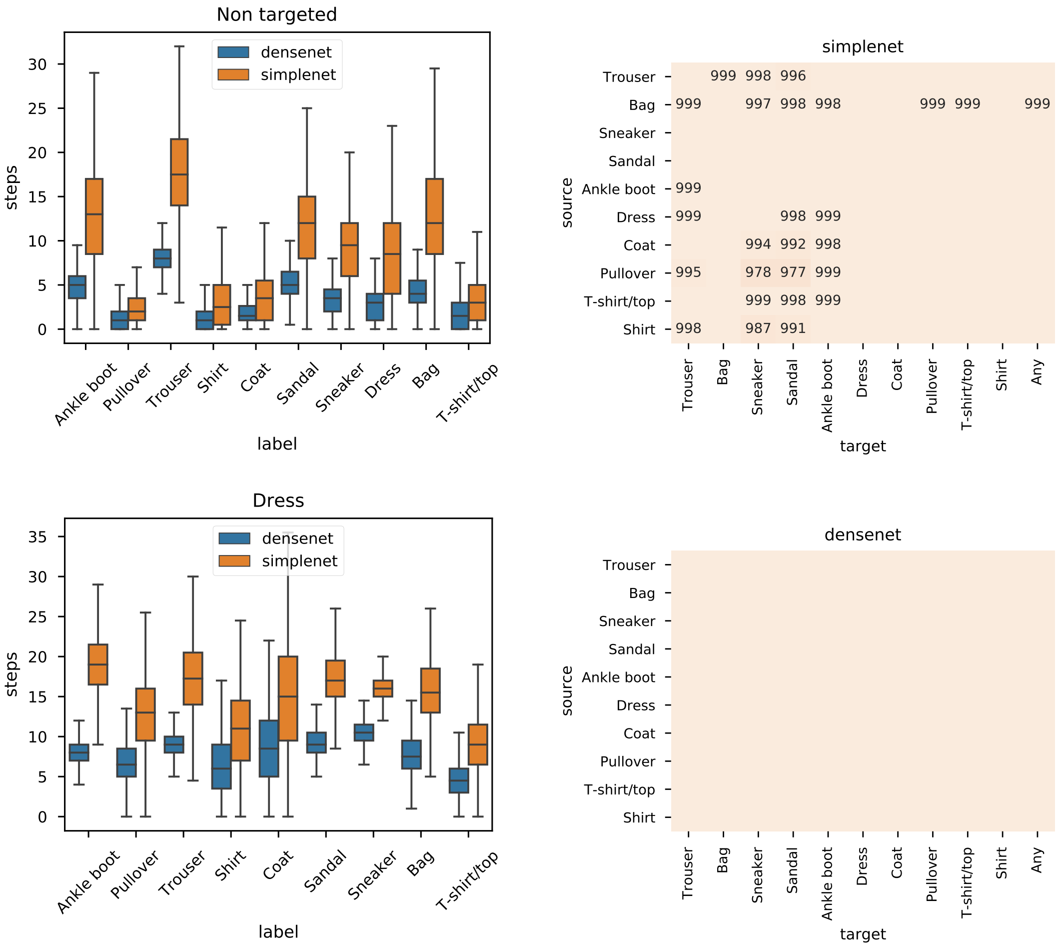

Figure 4.2: FGSM White-box

scenarios for each examined architecture. *Simplenet* architecture proves to be slightly more resilient both in terms of success rate and iterative FGSM step count. Consulting confusion matrix of targeted models, more confused classes are easier to attack in non-targeted scenario. In targeted attack the distribution of step count approximately corresponds to the distribution of targeted class confusion (column *dress* in confusion matrices).

Universal

4.1.2 Surrogate attack

Following experiment explores the transferability of adversarial examples using surrogate models. We compare the performance of following four scenarios.

plain surrogate model is model trained on the same dataset (not necessarily using the same train, validation splitting) 4.3

full surrogate model is trained to fit output probability distribution of targeted model using whole training dataset 4.4

reduced is similar to *full* surrogate, except for being trained only on a fraction of training data to mimic limited ability to call targeted model (when obtaining output probability distribution) 4.5

binary surrogate model is similar to *full* surrogate in terms of data size, but is trained as a binary classifier between one chosen class and all other classes, allowing for targeted attack to selected class and non-targeted attack from it 4.6

Appart from varying approaches towards surrogate training, we compare performance of surrogates being of the same architecture versus differing architecture from the targeted model. From attached plots (4.3, 4.4, 4.5, 4.6), we can see that

TODO turn bullet list into paragraphs??, evidence for feature reuse?

- *SimpleNet* is more resilient to attacks than *DenseNet* (left column of plots scoring lower values than right column), we assume that this might be caused by extensive feature reuse in case of *DenseNet* – tampering with some feature influence prediction of multiple classes (those dependent on the modified feature)
- it is easier to generate adversarial examples among visually similar classes – types of shoes (*sneaker, sandal, ankle boot*), clothes covering the chest (*dress, coat, pullover, t-shirt/top*), while class unlike any other (*trouser*) is resilient to adversarial attack; from this point of view, *bag* category might seem as an outlier, being very easily attacked – we think this might be again caused by feature reuse, as from the point of visual similarity, it is close to both shoes and pieces of clothes
- asymmetry in performance of cross architecture attacks is interesting phenomenon, with *DenseNet* target model attacked by *SimpleNet* surrogate being far more successful than its counterpart; same architecture attack, on the other hand proves to be successful in both cases, outperforming cross architecture attacks; this again may be caused by feature reuse with *DenseNet* with more categore independent features unable to attack *SimpleNet* with more category dependent features
- comparable performance of plain surrogates (trained on the whole dataset, unaware of target model distribution) and reduced surrogates (trained on reduced dataset, aware of target model) both inferior to full surrogates (trained on the whole dataset, aware of target model) which confirms that both increasing amount of data and increasing knowledge about targeted model improve succes rate of adversarial attack

TODO place plots before analysis? to backreference??
Experiments will be carried out for those scenarios:

- EA (black-box, pure EA)
- hybrid (surrogate, EA + pre-trained surrogate)
- hybrid (surrogate, EA + on-demand-trained surrogate)

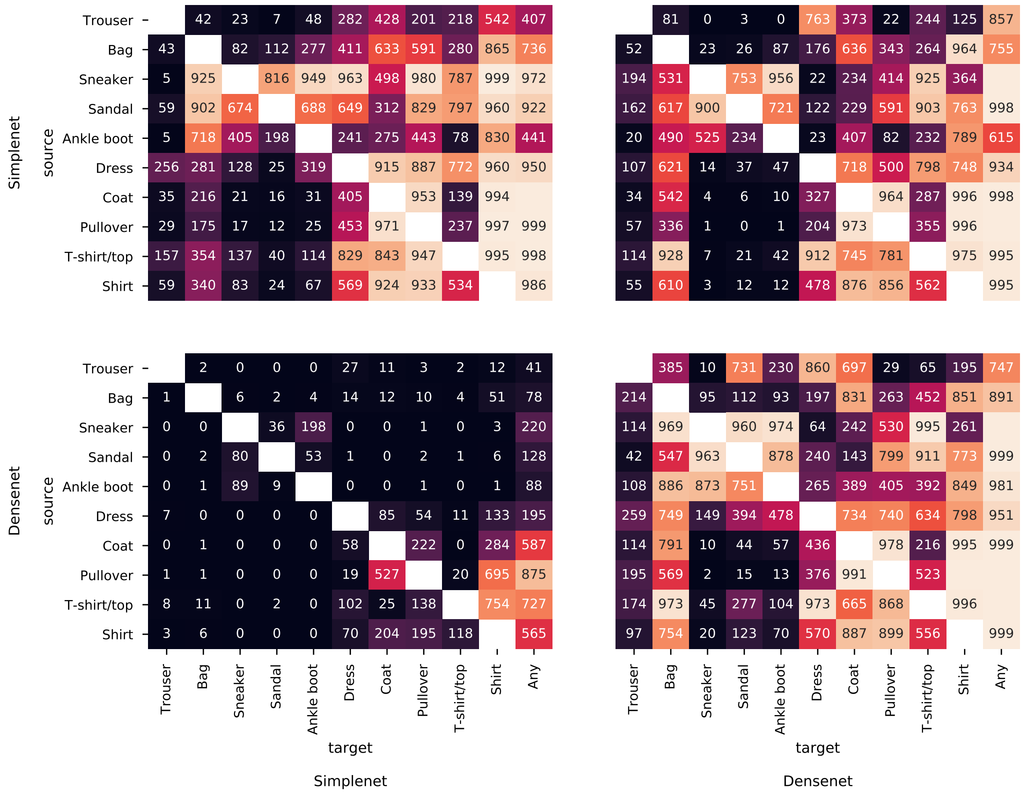

Figure 4.3: Plain surrogate success rate

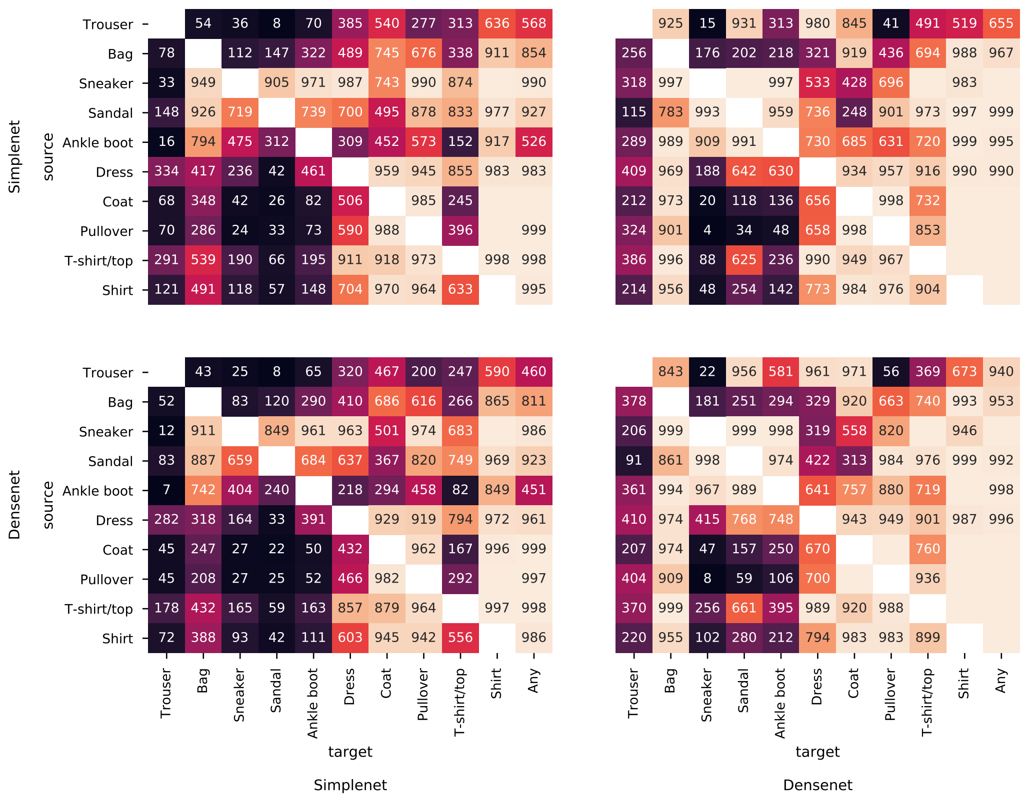

Figure 4.4: Full surrogate success rate

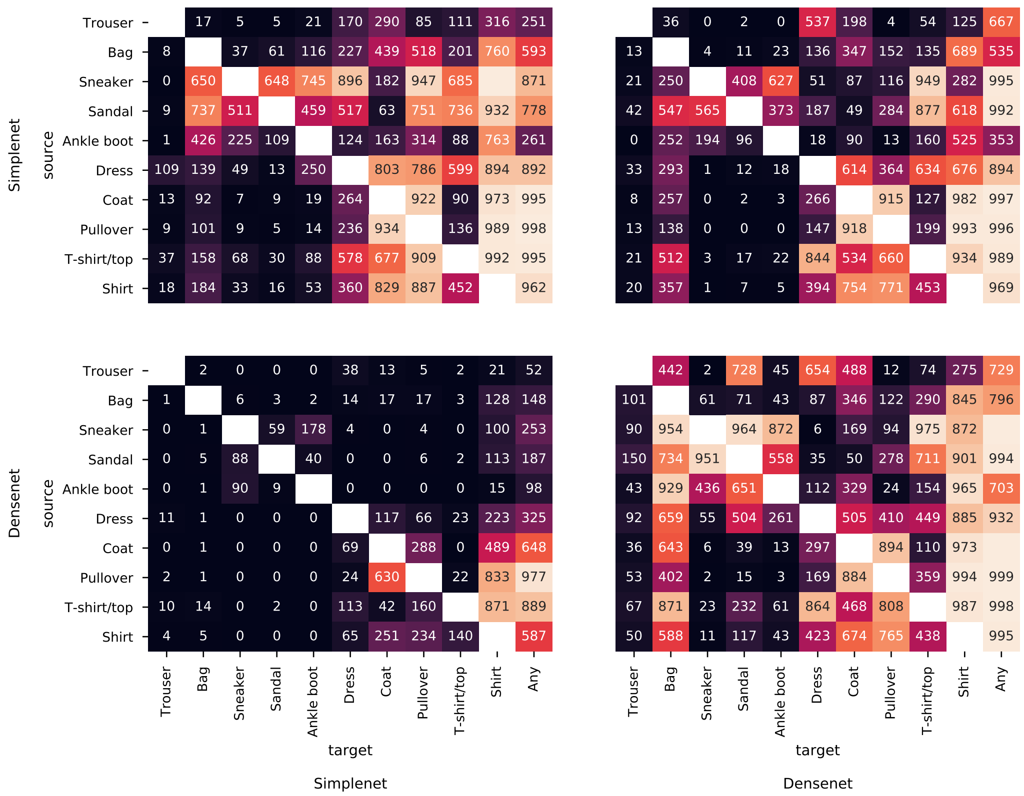

Figure 4.5: Reduced surrogate success rate

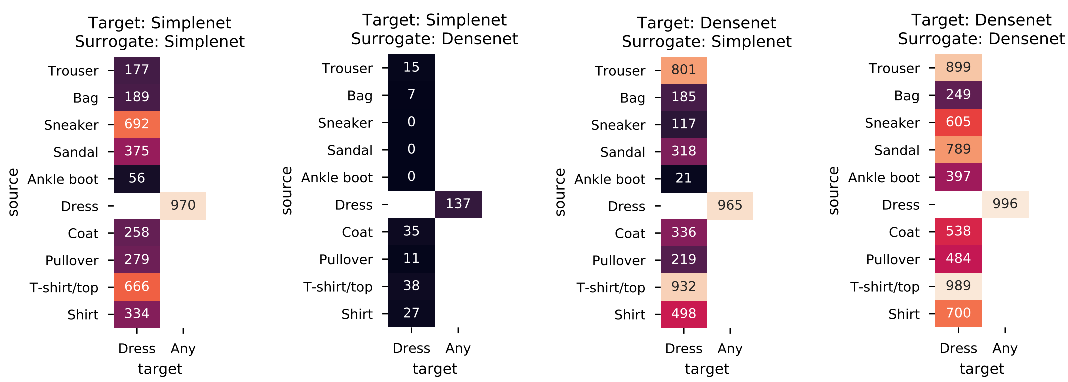

Figure 4.6: Binary surrogate success rate

Each combination of following options will be tested:

- targeted vs non-targeted
- single image vs multi-image vs generalization (unseen images)

Each experiment will be cross-validated using 10-fold cross-validation.

Architectures - Simple CNN - 3 cnn layers with max-pooling and one hidden dense layer - DenseNet-BC-8-52

Datasets - Fashion MNIST - Cifar-10 - bude-li čas

median simple_cnn model -¿ fold_7 median dense_net model -¿ fold_0

experiments on test set - unseen data - common scenario in real world FGSM white-box, target model - confidence bound 0.5, 0.95 (for surrogate) - whole test set, targeted/non-targeted, single-image - multi-image - whole class - outcomes (metrics) ?? - median/mean/std/quantiles of step count for iterative FGSM ? - median/mean/std/quantiles of added noise? (MSE?) - rate of success (will be close to 1) - time ? - some examples? which ones (unsuccessfull ones if any?) - show visual FGSM surrogate (different seed) - cross-validation folds cross comparison (whole/partial?) - 2 vs 20 (wiht 0.95 bound) - median model - median of accuracy - multi-image - whole class FGSM surrogate (teacher-student) - 2 median modely * 10 fold * 2 architectures - whole dataset (resplitted randomly) - 1/10th of dataset (resplitted randomly) - multi-image - whole class FGSM surrogate (binary teacher-student) - 2 median-model * 10 fold * 2 architectures * 2 big small data (just one median class) - (maybe if really bad - balance dataset) EA (black-box, pure EA) - 10 random (really random) samples from test set - multi-image - vs median models hybrid - pre-trained surrogate - 2 median-surrogates * 2 binary/full * 10 random sample * ... hybrid - on-demand surrogate - 2 binary/full * 10 random sample * 2 * policies

4.2 Evolutionary Algorithms

4.3 Hybrid methods

Conclusion

Bibliography

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. EMNIST: an extension of MNIST to handwritten letters. *CoRR*, abs/1702.05373, 2017. URL `http://arxiv.org/abs/1702.05373`.

G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989. ISSN 1435-568X. doi: 10.1007/BF02551274. URL `https://doi.org/10.1007/BF02551274`.

Andries P. Engelbrecht. *Computational Intelligence: An Introduction*. Wiley Publishing, 2nd edition, 2007. ISBN 0470035617.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL `http://arxiv.org/abs/1512.03385`.

Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016. URL `http://arxiv.org/abs/1608.06993`.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. URL `https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012a. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012b. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998. URL `http://yann.lecun.com/exdb/publis/pdf/lecun-98.pdf`. Last visited 2018-07-02.

George A. Miller. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41, November 1995. ISSN 0001-0782. doi: 10.1145/219717.219748. URL `http://doi.acm.org/10.1145/219717.219748`.

Tom M. Mitchell. *Machine Learning*. McGraw-Hill Education, 1997. URL `http://www.cs.cmu.edu/~tom/mlbook.html`.

National Institute of Standards and Technology. Nist special database 19, 2010. URL `https://www.nist.gov/srd/nist-special-database-19`. Last visited 2018-07-02.

Andras Rozsa, Ethan M. Rudd, and Terrance E. Boult. Adversarial diversity and hard positive generation. *CoRR*, abs/1605.01775, 2016. URL `http://arxiv.org/abs/1605.01775`.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. URL `http://arxiv.org/abs/1409.1556`.

Sho Sonoda and Noboru Murata. Neural network with unbounded activations is universal approximator. *CoRR*, abs/1505.03654, 2015. URL `http://arxiv.org/abs/1505.03654`.

Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *CoRR*, abs/1312.6199, 2013. URL `http://arxiv.org/abs/1312.6199`.

Zhou Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *Trans. Img. Proc.*, 13 (4):600–612, April 2004. ISSN 1057-7149. doi: 10.1109/TIP.2003.819861. URL `http://dx.doi.org/10.1109/TIP.2003.819861`.

Yuxin Wu and Kaiming He. Group normalization. *CoRR*, abs/1803.08494, 2018. URL `http://arxiv.org/abs/1803.08494`.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017a. URL `http://arxiv.org/abs/1708.07747`.

Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: Benchmark, 2017b. URL `https://github.com/zalandoresearch/fashion-mnist#benchmark`. Last visited 2018-07-03.

Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. *CoRR*, abs/1710.11342, 2017. URL `http://arxiv.org/abs/1710.11342`.

List of Figures

List of Tables

List of Abbreviations

A. Evgena Framework User Documentation

B. Attachments

B.1 First Attachment