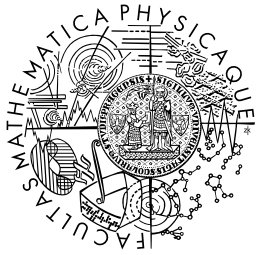

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

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: Machine learning models exhibit vulnerability to adversarial examples i.e., artificially created inputs that become misinterpreted. The goal of this work is to explore black-box adversarial attacks on deep networks performing image classification. The role of surrogate machine learning models for adversarial attacks is studied, and a special version of a genetic algorithm for generating adversarial examples is proposed. The efficiency of attacks is validated by a multitude of experiments with the Fashion MNIST data set. The experimental results verify the usability of our approach with surprisingly good performance in several cases, such as non-targeted attack on residual networks.

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. Those tasks form the field of artificial intelligence. Even though the artificial intelligence has undergone several crisis, not being able to deliver on its promises, it is experiencing boom once again thanks to advances in machine learning.

The ability to learn is believed to be the key of our success, the success of the mankind as a species, and accounts to what we call the intelligence. Following this belief, the machine learning mimics the process of learning by automating extraction of knowledge from experience, with the aim to abstract and grasp the semantics of the task. This is reached by construction of mathematical functions, which for given, numericaly represented, observations return valid, numericaly represented, conclusions. Those functions often take form of composite paramteric functions, and to solve the task satisfactorily, the best possible composition – architecture, together with the best possible set of parameters is looked for. The deep learning is current state of the art method of machine learning.

The deep learning is based on use of deep neural networks. Neural network is a structure of interconnected artificial neurons, with artificial neuron being unit computing non-linear transformation of linear combination of its inputs, passing this value further to other connected units. With increasing size of those networks in terms of unit count, the convenient abstraction of layers – mutually dependent groups of mutually independent units, emerged. Early neural network based solutions used very few of those layers, mainly due to the hardware limitations, but with the advances in technology, the deep learning i.e., usage of many-layered neural networks, was established. Appart from rising complexity of the models, new types of layers were invented, namely the convolutional layers extracting the local context from observations. Recurrent neural networks enabling processing of sequences of data were invented as well.

It was the convolutional neural networks i.e., the deep neural networks using convolutional layers, that lift off the new era of image processing, part of the field of computer vision. One of the key tasks solved by computer vision techniques is the task of image classification, which lies in assigning a category from the set of possible categories, to the image. The subjects of those task cover a wide range, from classifying real world entities on photographs to medical applications and deciding if the PET/CT scan reports cancerous formations. Use cases with high demands on reliability, like security enforcing systems, or aforementioned medical applications, need to be resilient to mistakenly corrupted data or even targeted attacks.

Deep learning image classification solutions reports vulnerability to adversarial examples i.e., artificially created inputs that become misinterpreted. It has

been showed that carefully crafted perturbations added to images may successfully lead to model failure, while being unnoticable for the human observer, or a different model. The issue of vulnerability to adversarial attacks does not apply only to the task of image classification, but it also concerns speech recognition, particularly use of virutal personal asistants; or image segmentation and object detection on which autonomous driving depends. It is of great importance to study those vulnerabilities, attacks and corresponding defences to improve reliability of such solutions.

The goal of this work is to explore adversarial attacks in image classification by deep networks in the black-box scenario. We want to assess the suitability of genetic algorithms as a generative procedure for finding adversarial examples. In order to compare the proposed strategy, a careful analysis of utilization of surrogate models in the context of generation of adversarial examples needs to be performed. The aim is to train multiple deep neural networks, both as target and surrogate models, of varying architectures in several contexts, such as targeted and non-targeted attacks, or complete vs. one-vs-all (binary) classification. The performance of aforementioned approaches will be compared with state of the art white-box attacks represented by the fast gradient sign method.

The structure of the thesis is as follows. In chapter 1 the task of image classification is defined, along with overview of available datasets for research purposes (section 1.1). The underlying theory of deep learning for the purpose of image classification (section 1.2), along with theoretical background of genetic algorithms (section 1.3), is presented. Chapter 2 spans the definition of adversarial attacks, their taxonomy (section 2.1) and known attack strategies (sections 2.2 and 2.3). In chapter 3, we propose our solution (section 3.2). In chapter 4, we analyse known aproaches (sections 4.1.1 and 4.1.2) and compare them to our solution (section 4.1.3), carrying out a multitude of experiments.

1. Preliminaries

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 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 (CV), 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 an 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 graphics processing unit (GPU) accelerated computation. See Szeliski [2010] for in depth introduction to the field of computer vision.

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 section 1.2). Those models outperformed former state of the art methods i.e., support vector machines with RBF kernels or scale-invariant feature transform 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 convolutional neural networks 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 $\mathcal{I}$ 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 (see Szeliski [2010]), or by building a multitude of solutions – each targeted at specific shape of input data. Several derived tasks e.g., *hierarchical classification* with labels forming tree-like structure, or *multilabel classification* with examples belonging to multiple categories exist, but are out of scope of this writing.

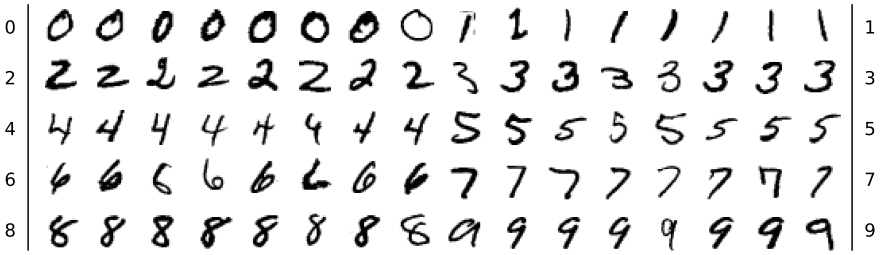

Figure 1.1: MNIST samples

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 $28 \times 28 \times 1$ pixels (px) (rectangular grayscale image) and often the same dataset 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 the following steps – each character is reshaped to fit $20 \times 20\,px$ patch preserving the aspect ratio and the center of mass of this patch is aligned with the center of larger $28 \times 28\,px$ patch creating final sample (see fig 1.1). 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 has become obsolete and is no longer a valid benchmarking dataset for performance comparison nowadays.

EMNIST The *Extended MNIST* (EMNIST) dataset contains alphanumeric characters and it 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 EMNIST is not a superset of MNIST. 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 (see Cohen

Figure 1.2: Fashion MNIST samples

et al. [2017]). Despite being substantialy harder task than MNIST, EMNIST is not widely used.

Fashion MNIST The *Fashion MNIST* by Xiao et al. [2017a] shares the layout of MNIST dataset, having the same number of same shaped train and test examples as well as the 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* and *Ankle boot* (see fig 1.2). This dataset seems to be a promising replacement of the original MNIST as it is defining harder and more computer vision relevant task (images of real world objects rather than man-made symbols), while preserving moderate hardware requirements and model complexity demands.

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* and *truck* (see fig 1.3), while CIFAR-100 introduces hierarchical system of 20 superclasses e.g., *fish, small mammals, tress, large carnivores*, each subdivided into 5 subclasses 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 (challenge based on ImageNet

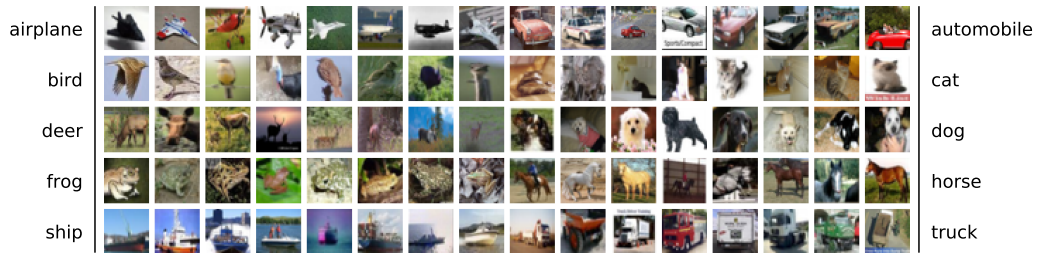

Figure 1.3: CIFAR 10 samples

dataset, see Russakovsky et al. [2015]) – 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 (see Zoph et al. [2017]).

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. See Goodfellow et al. [2016] for in depth introduction into the deep learning.

1.2.1 Machine Learning

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.

> 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])

More formally, the machine learning solution, often referred to as model M, can 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, often referred to as *weights* of the model. 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 $\bar{P}_{M_\theta}$ matches the probability distribution of random variable $\mathtt{x}$ (see Flach [2012]). This process is implemented as a minimization of loss function (see section 1.2.2), often using gradient descent algorithm (section 1.2.3).

Given the characteristics of the training data, machine learning tasks can be split into the following two categories – *supervised* and *unsuperivsed*.

Training data of the *supervised* task take 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.

Training data of the *unsupervised* task take form of singletons x i.e., samples of unknown distribution. The goal is usually to find an 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 referred 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.

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

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 note that KL-divergence is not symmetric, hence it is not a distance measure.

Mean squared error The *mean-squared error* (MSE) loss function is widely used in regression tasks as it meassures the sum of variance and squared bias of the model error. It takes the following form.

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

where n is the number of samples.

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 (see section 1.2.4). 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. 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 ∇_θ 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* (SGD) for we have access only to the approximation of the loss function in each step.

Algorithm 1 Stochastic gradient gescent [optionally with Nesterov momentum]

$\alpha \leftarrow$ learning rate
$[\beta \leftarrow$ momentum rate$, v \leftarrow 0]$

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

The SGD with learning rate α and *Nesterov momentum* of rate β (algorithm 1) runs as follows. We set the initial momentum to zero, then each step, until we have reached the goal, we first change parameters according to the accumulated momentum, we estimate the gradient of the loss function, then we update the momentum and perform step according to the estimated gradient.

Adam The *Adam* optimizer by Kingma and Ba [2014] is one of the optimizers derived from basic SGD. 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 (algorithm 2) 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.

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
  Sample a minibatch of m training examples $\left(x^{(i)}, y^{(i)}\right)$
  $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L\left(M_\theta\left(x^{(i)}\right), y^{(i)}\right)$
  $t \leftarrow t + 1$
  $s \leftarrow \beta_1 s + (1 - \beta_1)\, g$
  $r \leftarrow \beta_2 r + (1 - \beta_2)\, g^2$
  $\hat{s} \leftarrow s / (1 - \beta_1^t)$
  $\hat{r} \leftarrow r / (1 - \beta_2^t)$
  $\theta \leftarrow \theta - \frac{\alpha}{\sqrt{\hat{r}} + \epsilon} \hat{s}$
until early stopping criterion is met

1.2.4 Regularization

As stated before, the issue of overfitting is adressed by means of *regularization* – methods, that improve model generalization ability (performance on unseen data), while not neccessarily changing the performance on training data. Frequently used regularization algorithms are: *early stopping, p-norm regularization, dataset augmentation, ensembling, dropout*, etc. See Goodfellow et al. [2016] for in depth overview.

The *early stopping* is a policy, terminating training process if model performance on validation set visibly stalls or worsens.

The *p-norm regularization* adds additional term to the loss function computing *p-norm* $||\cdot||_p$ of model parameters θ.

$$||\theta||_p = \sqrt[p]{\sum_{t \in \theta} t^p} \tag{1.3}$$

This forces the parameters to maintain mean close to zero and low variance; $l2$ and $l1$ norms are often used.

The *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 and addition of random noise are broadly used augmentation methods. They need to be tailored to task specifically, to preserve label of augmented example.

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

The *dropout* masks internal featuremaps (see section 1.2.5) randomly during the training process, 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. We will refer to tensors being passed between layers as *featuremaps* or *hidden representation*, and slices of featuremaps along the last axis will be called *channels*. See Goodfellow et al. [2016] for more.

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 (1.4).

$$L_{W,b}\left(x_{in}\right) = Wx_{in} + b \tag{1.4}$$

Activation functions Linear combinations itself are not able to form strong enough models. Cybenko [1989] proved, that added non-linearity, in their case *sigmoid* function (1.5), 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* (1.6) and its modifications, etc. (see Sonoda and Murata [2015]). Those are widely used in machine learning as an *activation functions*.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{1.5}$$

$$f(x) = \begin{cases} x, & \text{for } x > 0 \\ 0, & \text{for } x \leq 0 \end{cases} \tag{1.6}$$

Activation function as a neural net layer is then a straightworward elementwise application of it to the input featuremap.

Convolution A multitude of machine learning tasks process data with inherent spatial properties – mainly context locality (e.g., images, natural language samples or audio). *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 two-dimensional convolution (Conv2D) 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 (1.7).

$$\text{Conv2D}(X_{in})_{x,y} = \sum_{i=1}^{w_K} \sum_{j=1}^{h_K} \left(W_{i,j} X_{in_{x+i-\lceil w_K/2 \rceil, y+j-\lceil h_K/2 \rceil}} \right) + b \tag{1.7}$$

As stated in (1.7), each output value $X_{out_{x,y}}$ is computed as a sum of the elemntwise multiplication of convolution kernel with equally shaped part of input featuremap centered around the position corresponding to $X_{out_{x,y}}$. N-dimensional convolution for different n can be computed analogically. As the convolution crops the featuremap proportionally to the size of the kernel, *padding* of the input tensor

may be used as a preprocessing step, to maintain constant hidden representation size. To perform featuremap downsampling, we can compute convolution only in evenly spaced positions, with the distance between those positions being referred to as *stride*.

Pooling *Pooling* represents another way of downsampling. Unlike strided convolution, pooling is often parameterless and instead of performing convolution with trainable kernel, it applies given function to sliding window (channel-wise). *Maximum* and *average* are frequently used pooling functions – layers are then called *max-pooling* and *avg-pooling*, respectively.

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 statistical properties of outputs of preceding layers. *Normalization layer* keeps track of the first and the second moment estimate often using exponential moving average. Normalization yields moderate regularization effect.

There is a multitude of normalization layers varying in a way those moments are computed. According to the span of normalization, we recognize three types of it: *batch, layer* and *group*.

The *batch normalization* takes place over all but last axis (channels) of featuremap. The gain of employing bacth normalization depends on training batch size, as bigger batches benefit more (see Ioffe and Szegedy [2015]). The *layer normalization* takes place over all but first axis (examples) of featuremap. Unlike batch normalization, the performance is not compromised when using smaller batch size. Finally, the *group normalization* takes place over multiple channels per example, forming multiple groups of features. This effectively forces the 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 deep convolutional neural networks, followed by gradually deeper architectures *VGGNet, ResNet, DenseNet*, etc. Apart from image classification, deep learning models dominated image segmentation and natural language processing tasks e.g., machine translation, speech recognition, synthesis, etc. Architectures for those tasks are out of scope of this writing.

AlexNet *AlexNet* by Krizhevsky et al. [2012b] is a convolutional neural network using convolutions of size $11 \times 11, 5 \times 5$ and 3×3, max-pooling for downsampling, $ReLU$ as an activation function and dropout after dense layers for regularization. It has 8 layers consisting of 62.3 million trainable parameters. SGD with momen-

tum is used for training. The *AlexNet* scored 15.3% top-5 accuracy in ILSVRC 2012 competition.

VGGNet With twice as many layers as *AlexNet* and more then double parameter count, *VGGNet* by Simonyan and Zisserman [2014] is another well known architecture. It uses exclusively 3×3 convolutions, max-pooling for downsampling, $ReLU$ as an activation function and dropout for regularization after dense layers. SGD with momentum is used for training. With 16 layers and 138 million parameters, *VGGNet* scored top-5 accuracy of 7.3%.

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, their *ResNet-152* model, with 152 layers and parameter count of 60.2 millions, scored top-5 accuracy of 3.6%. It uses 3×3 convolutions, average pooling, $ReLU$ activation and dropout after dense layers. Moreover batch normalization between each convolution and activation is used. It is trained using SGD with momentum.

DenseNet Further exploiting the performance gain of residual shortcuts usage, Huang et al. [2016] created the *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 Genetic Algorithms as Space Search Algorithm

Genetic algorithms represent one of heuristic search algorithms, and they are inspired by the process of natural evolution. Following the work of Darwin [1859], 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 a population P is a subset of space to be searched ($P \subset S$), with individual i being an element of population ($i \in P$), hence element of search space S i.e., representing a possible solution. Fitness function $F : i \mapsto \mathbb{R}$ is a real valued meassure of fitness depending on individual, often function of the objective function O of the task to be solved i.e., $F : O(i) \mapsto \mathbb{R}$. As an operator $\mathrm{Op} : P_{in_1}, P_{in_2}, \ldots, P_{in_n} \mapsto P_{out}$ we define

a function converting one or more populations to a new one, often in element-wise (per individual) manner. *Epoch* is then one step of the genetic algorithm consisting of application of the sequence of operators to current population P_t, producing new population $P_{t+1} = \mathrm{Op}_n\left(\mathrm{Op}_{n-1}\left(\ldots \mathrm{Op}_1(P_t)\right)\right)$.

The run of the genetic algorithm begins with *initialization* i.e., creation of the initial population. Frequent approaches towards initialization are sampling of chosen probability distribution or usage of predefined set of promising individuals – gene pool, often obtained by some task specific heuristics. After the initialization, the main loop is entered. The genetic algorithm stops if the stopping criterion is met i.e., if some individual of the current population, or the whole population itself, reaches sufficient performance with respect to objective O.

It is worth to note that population and consequently individuals are often not sampled from space S directly but rather conveniently *encoded* in a format suitable for genetic algorithm. The representation of individual often takes form of an array of features, often referred to as *genome* with individual features called *traits* or *genes*.

1.3.1 Operators

The ability of genetic 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* which combines several individuals into new one; *mutation* operators, which 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.

Assuming the population P of individuals being vectors of genes $i \in G_1 \times G_2 \times \cdots \times G_l$ of length l, with G_k being the *domain* of gene g_k. The following paragraphs elaborate on several most frequently used operators.

N-point cross-over 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 often used.

In case of *random mutation*, 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*); in case of real valued genes, normal distribution centered in g_k with given variance (*biased mutation*), etc.

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 chosen

multiple times, small populations suffer from high variance in roulette wheel sampling. This issue is addressed by stochastic universal sampling algorithm (see Engelbrecht [2007]).

Another stochastic selection operator is the *tournament selection*. Given target size of output population s_{out} and tournament size s_t, the output population is selected by running s_{out} independent tournaments i.e., taking the fittest individual (winner) from random input population sample of size s_t (contestants). In case of fitness function not corresponding to real fitness of the individual in terms of proportionality and scaling, tournament selection outperforms roulette wheel selection, as it depends only on the ordering of the fitness of the individuals.

2. Adversarial Examples for Deep Learning Models

Performance of deep learning models is not perfect and certain amount of input data gets missclassified 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 and 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 the following, mutually independent properties i.e., *specificity, scope, adversary's knowledge* and *perturbation constraints*.

The *specifity* of adversarial attack is defined by the target class l'. In case of *targeted* attack, the target class l' corresponds to one specific classification category. On the other hand, in case of *non-targeted* attack, l' corresponds to all but the ground truth class l and adversarial example is than any $x' = x + \eta$ such that $M_\theta(x') \neq l$.

The *scope* of adversarial attack lies in number of original examples to be attacked by single η. *Singleton* attack stands for attack on single example x such that $M_\theta(x + \eta) = l'$. *Multi* attack stands for attack on arbitrary number of examples X of the same class such that $\forall x \in X : M_\theta(x + \eta) = l'$. Finally, *universal* attack seeks single η such that $\forall x : M_\theta(x) = l \implies M_\theta(x + \eta) = l'$

We distinguish between three cases of *adversary's knowledge*. The *white-box attack* assumes full knowledge of targeted model i.e., architecture, weights and training data as well as parameters of the optimizer being used for training. On the other hand, the *black-box attack* assumes no additional information about targeted model, appart from the classification predictions and respective probabilities. Finally, the *surrogate attack* treats the targeted model as a black-box, yet has full access to substitute model i.e., model sharing some characteristics of target model e.g., task it solves, training data, architecture, output probability distribution, etc.

Regarding the *perturbation constraints* of the attack, we distinguish the following three cases. *Unconstrained* attack with no limitations on perturbation η, *constrained* attack, with limited intensity of perturbation η allowed, with respect to some measure, and finally the *optimized* one, where the perturbation intensity is minimized. The $l1$ and $l2$ norms, or *PASS* by Rozsa et al. [2016] and *SSIM* by Wang et al. [2004] – visual similarity measures, are frequently used intensity measures.

2.2 Gradient Based Methods

Thanks to the 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 the *L-BFGS* using Broyden–Fletcher–Goldfarb–Shanno optimization algorithm, fast gradient sign method and its modifications, feature adversary, one-pixel attack, etc. See Yuan et al. [2017] for overview of adversarial attack approaches.

Fast Gradient Sign Method The fast gradient sign method (FGSM) computes gradients of loss function of the target model with respect to the target example x and class l'. The sign 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 substraction, we use addition to move the target image in direction of falling source class prediction.

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

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 (generator G) on target model task dataset, mapping random latent space to images and invertor I, mapping images to latent space of generator G. The adversarial examples are generated by finding z' close to $z = I(x)$, such that $M_\theta\left(G(z')\right) = l'$. This approach genererates adversarial examples more natural to human observer as we are perturbing the hidden representation rather than the final output.

3. Our Solution

In our solution we focus on design of a black-box adversarial attack, for this is the likely real life scenario, as the number of web services offering image classification is soaring and image classificators are being embedded into various devices e.g., security cameras. We will shortly elaborate on available black-box attack strategies and then we will present and discuss our pure genetic algorithm based approach.

3.1 Related Work

Regarding the black-box adversarial attacks for deep learning models, only a few methods has been proposed so far. Papernot et al. [2016] suggest an approach converting black-box attack to surrogate attack, by synthesizing dataset for the surrogate model to be trained on. Narodytska and Kasiviswanathan [2017] perform local-search of the neighbourhood of targeted image to estimate the gradient of targeted model. This estimate is later use to guide the generation of adversarial example. The former approach is later extended by Ilyas et al. [2018], who employ natural evolutionary strategies to estimate target model gradients with respect to targeted image. The adversarial example is generated using the gradient descent algorithm on the estimated gradient. Finally, Vidnerová and Neruda [2016] used genetic algorithms to perform adversarial attack mostly on shallow machine learning models. We will base our solution on their approach, evaluating the method on the state of the art deep learning models and proposing a multitude of improvements to the genetic algorithm based strategy.

3.2 Evolutionary Generated Adversarial Examples

Analysing approaches towards generation of adversarial examples in black-box scenarios, we chose evolutionary algorithms – particularly genetic algorithms, as promising candidates. Unlike other heuristic random space search algorithms, genetic algorithms exploits the inherent property of the way that deep learning models process imagery data. Using reproduction operators, individuals in population interact with each other, allowing for recombination of promising genes. As the deep convolutional neural networks extract the information from the local context using convolutional layers, we assume that by combining parts of promising adversarial examples, we can obtain ones scoring even better results.

As already stated, the method we propose is based on use of genetic algorithms by Vidnerová and Neruda [2016]. They propose the following configuration of the solution. For the targeted model M_θ, with input $[0,1]^{h\times w}$, the individual i is encoded as a vector of pixel intensities ($i \in [0,1]^{hw}$). The two-point crossover is selected as a reproduction operator, and the biased random mutation with Gaussian distribution is employed. Selection is performed using tournament selection of size 3 and relies on fitness function which is an arithmetical average of $l2$ norm of perturbation intensity and $l2$ distance between prediction of individual

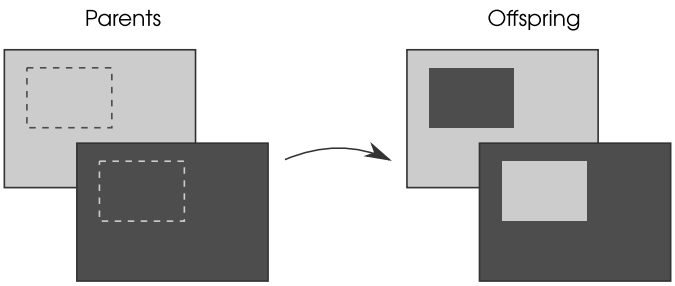

Figure 3.1: Two-point cross-over on images

under M_θ and targeted prediction. The evaluation, performed on several shallow models trained on MNIST dataset, runs 10 000 epochs of genetic algorithm with population size of 50, allowing for at most 500 000 accesses to the targeted model (see section 1.3 on genetic algorithms theory).

In our solution, we propose the change of encoding of the individual to *2D* matrix to maintain spatial relations of the data – images. Those relations are further exploited by selected operators, namely cross-over. The suitability of this change is supported by the fact, that vast majority of deep learning image classifiers is based on use of convolutional layers, which extracts the spatial context of processed data. We have also experimented with generating perturbations smaller in terms of shape, which were added to the targeted image using bilinear interpolation. While proving moderatly successful, they did not outperformed full sized scenario. Our individual is thus defined as a matrix $i \in [0,1]^{h \times w}$ and represents perturbation η. We use zero centered binomial distribution scaled by factor of $\frac{1}{255}$ to sample initial population.

The use of two point cross-over is preserved with necessary changes to support the individuals of higher dimensionality. In our case the cross-over swaps randomly selected rectangular regions i.e., image crops of parent individuals, to produce offspring (see fig 3.1). We experimented with the use of universal crossover without success – we assume, that the degradation in performance is again caused by the inherent properties of convolutional layers, where the partial change of area covered by convolutional kernel yields unstable results in convolutional layer activation, while the change of whole rectangular region maintains the expected local response of the layer.

The idea of biased mutation is preserved as well, but we change the probability distribution to binomial distribution i.e., discrete probability distribution maintaining the discrete characteristics of intensity of pixels. As a result, biased mutation driven by zero centered binomial distribution mimics shifting of the intensity of the pixel by several shades of gray to either direction, while at the same time maintaining the beneficial properties of normal distribution. The binomial mutation (Mut) of individual i, with binomial distribution $\mathcal{B}(n,p)$, has the form of (3.1).

$$\mathrm{Mut}(i)_{x,y} = \begin{cases} i_{x,y} + \frac{b}{255}, & \text{with probability } p_{gene}, \text{ where } b \sim \mathcal{B}(n,p) - \frac{n}{2} \\ i_{x,y}, & \text{with probability } 1 - p_{gene} \end{cases} \tag{3.1}$$

For the optimization we use two objectives – the score of prediction of targeted

class, which needs to overcome 0.5 for successful attack; and $l2$ norm of perturbation (individual) $\eta = i$ (3.2), to quantify the intensity of it. We assume $l2$ norm being promising option for it accounts more for big pixel-wise differences. Appart from that, we suggest the use of clipping on the intensity of perturbation if it gets below certain admissible value c_{bound} i.e., taking the $\max(c_{\text{bound}}, l2(i))$ as a perturbation intensity, combining the *constrained* and *optimized* adversarial attack paradigm.

$$l2(i) = \sqrt{\sum_x \sum_y i_{x,y}^2} \tag{3.2}$$

As a result, the selection operator is changed to selection suitable for multi-objective optimization, namely non-dominated sorting algorithm. For that matter, we choose *NSGA-II* with crowding distance, which uses concept of Pareto-optimality to construct a so called *non-dominated fronts* – groups of mutually non-dominated individuals. Furthermore NSGA-II employs secondary sorting algorithm (in our case crowding distance) allowing for more stable selection of individuals (see Deb et al. [2002] for more). We assume this approach superior to average of objectives for the issues with different scaling of prediction error and perturbation intensity (see algorithm 3). The new population is selected from the union of parents and offspring of the current generation.

Algorithm 3 NSGA operator

$P_{in} \leftarrow$ input population
$s_{out} \leftarrow$ target size of selected population

$P_{out} \leftarrow$ empty list
while $|P_{out}| < s_{out}$ **do**
  $f \leftarrow$ pop non dominated individuals from P
  if $|P_{out}| + |f| <= s_{out}$ **then**
    extend P_{out} with f
  else
    sort f by crowding distance
    extend P_{out} with first $s_{out} - |P_{out}|$ elements of f
  end if
end while
return P_{out}

Finally, the dual clipping takes place before evaluation of objectives. Firstly, to maintain the individual in the range of a unit cube (the searched space) and secondly to maintain the pixel-wise perturbation in the allowed range – this is employed to reject individuals with few significantly shifted pixels, as those do not show sufficient visual similarity to targeted image. Clipping (Clip) with lower bound l and upper bound u for individual i corresponds to (3.3), clipping of the whole population clips each individual.

$$\text{Clip}_{(l,u)}(i)_{x,y} = \max\left(l, \min(u, i_{x,y})\right) \tag{3.3}$$

The whole proposed strategy towards generating adversarial examples takes form of algorithm 4. We provide our implementation as an open-sourced project

Algorithm 4 Evolutionary generated adversarial examples

$M_\theta \leftarrow$ targeted model
$x \leftarrow$ targeted image, of shape $h \times w$, $l' \leftarrow$ targeted class
$s_{pop} \leftarrow$ population size, $n_{epoch} \leftarrow$ epoch count upper bound
$c_{\text{bound}} \leftarrow$ perturbation intensity clipping value
$\Delta_{max} \leftarrow$ max pixel-wise difference bound

$P \leftarrow \left[\frac{1}{255}\left(\mathcal{B}(n,p) - \frac{n}{2}\right)\right]^{h \times w}$
while not $\exists i \in P : M_\theta(i) = l'$ **do**
  $P_{old} \leftarrow P$
  shuffle P
  $P \leftarrow \text{Xover}(P)$
  $P \leftarrow \text{Mut}(P)$
  $P \leftarrow \text{Clip}_{(-\Delta_{max}, \Delta_{max})}(P)$
  objectives evaluation $\forall i \in P : \left(M_\theta\left(\text{Clip}_{(0,1)}\left(x + i\right)\right)[l'], \max\left(c_{\text{bound}}, l2\left(i\right)\right)\right)$
  $P = \text{NSGA}\left([P, P_{old}], s_{pop}\right)$
end while

under MIT license – see Štěpán Procházka [2018] github repository.

We evaluate our solution on the Fashion MNIST dataset, allowing each run of genetic algorithm to perform at most $\sim 65\,000$ evaluations of targeted model M_θ, we choose this amount to get comparable results with white-box and surrogate approaches trained on training set of Fashion MNIST of size $60\,000$. See section 4.1.3 for results and comparison.

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* and *DenseNet*.

SimpleNet is a 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 (see section 1.2.3) with default (recommended) parameters for 128 epochs. Initial learning rate 0.001 is divided by 5 in 50% and 75% of training process. Whole model has approximately 900k trainable parameters. 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 referring to them.

As a second architecture, we use *DenseNet-BC* of depth 52 with growth rate $k = 8$ and 0.5 compression factor specifically, 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 section 1.2.6).

Both models use weight decay of 0.0001 as an 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 fig 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. We can see that the class *shirt* gets missclassified the most as it scores lowest accuracy of $\sim 83\%$, often confused with *T-shirt/top* ($\sim 6\%$), then *coat, pullover* and *dress* (error rate of $2 - 5\%$). Appart from that, to some degree, the confusion matrices report symmetry – showing that most pairs of classes are confused mutually, if confused at all. Detailed observation hints the correspondence between confusion and visual similarity e.g., footwear or clothing classes.

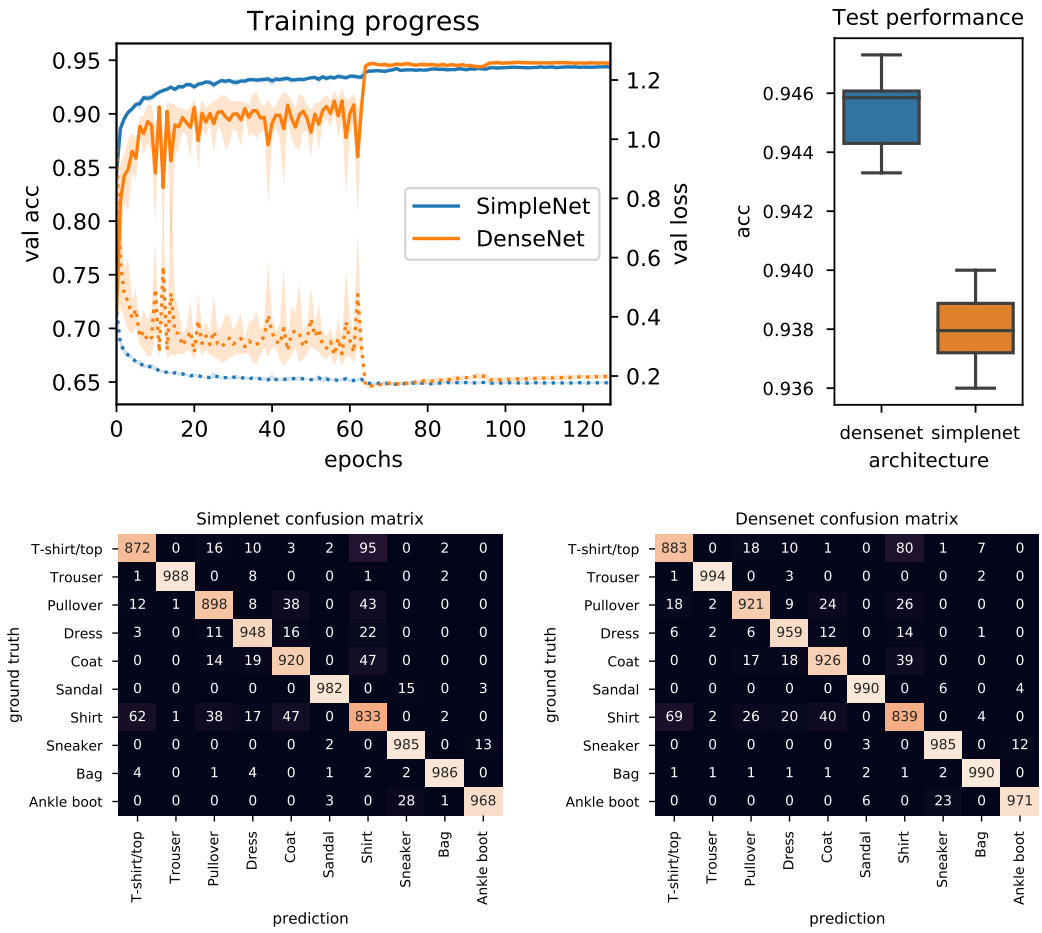

Figure 4.1: Cross validated training results

Experiments overview In the following sections we are going to examine following approaches towards generating adversarial examples for image classifiers and we will present their performance on the Fashion MNIST dataset.

Firstly, white-box fast gradient sign method attack serving as both implementation sanity check and baseline is carried out (see section 4.1.1). Next the transferability of adversarial examples is examined using surrogate attacks (see section 4.1.2). Those approaches include transfering adversarial examples between models trained on the same dataset – the most naive approach, training surrogate model on outputs of the 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, genetic algorithms are employed as a space search method, generating adversarial examples for the target model (see section 4.1.3).

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 (see fig 4.1).

Each approach is tested with either non-targeted and targeted singleton attack to each class. The attacks caried out are 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

In this section of experiments, the iterative FGSM (see section 2.2) 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 characteristics 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 then 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.

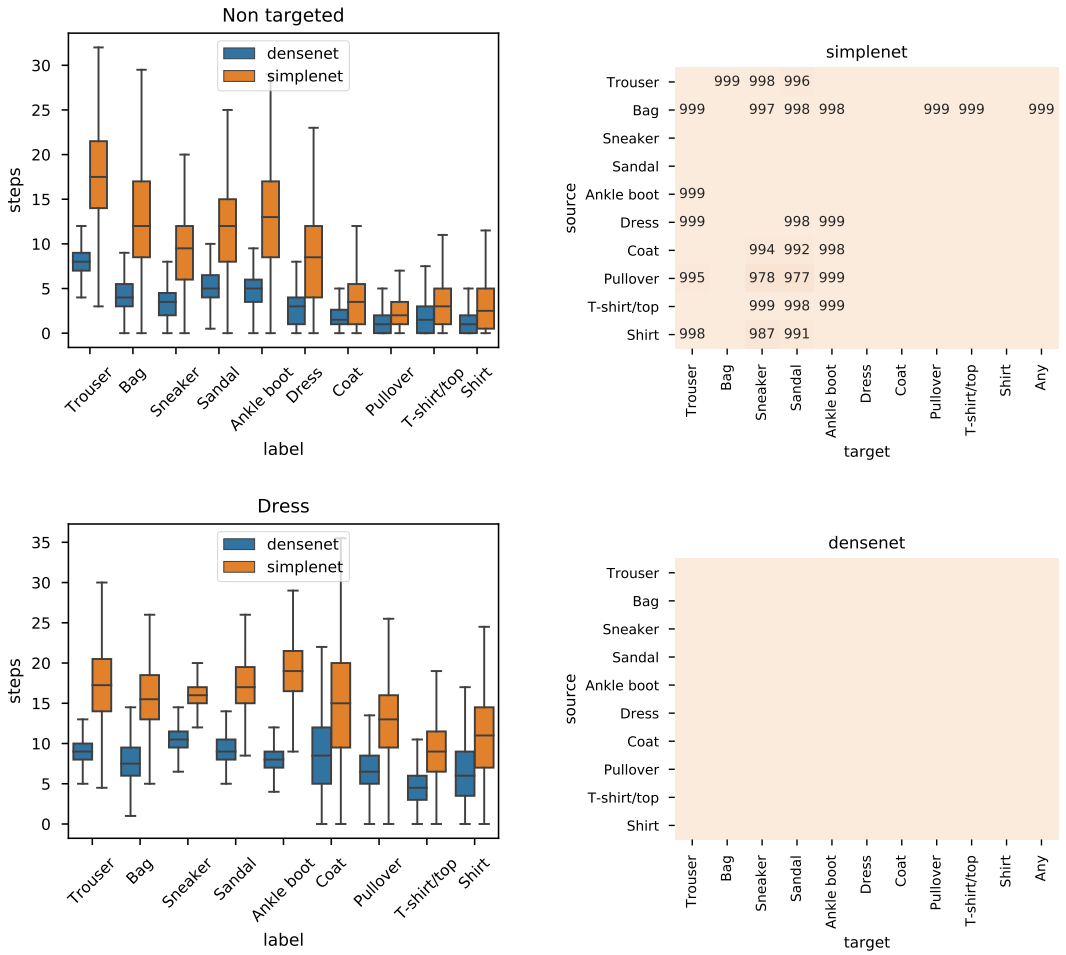

Figure 4.2: FGSM White-box

Results 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 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).

4.1.2 Surrogate Attacks

Following experiment explores the transferability of adversarial examples using surrogate models. We compare the performance of following four scenarios: *plain, full, reduced* and *binary*.

Plain surrogate model is trained on the same dataset (not necessarily using the same train, validation splitting) as the target model. *Full surrogate* model is trained to fit output probability distribution of targeted model using the response of it on the whole training dataset. *Reduced surrogate* is similar to full surrogate, except for being trained only on a fraction of training data to mimic limited ability to evaluate targeted model (when obtaining output probability distribution). We

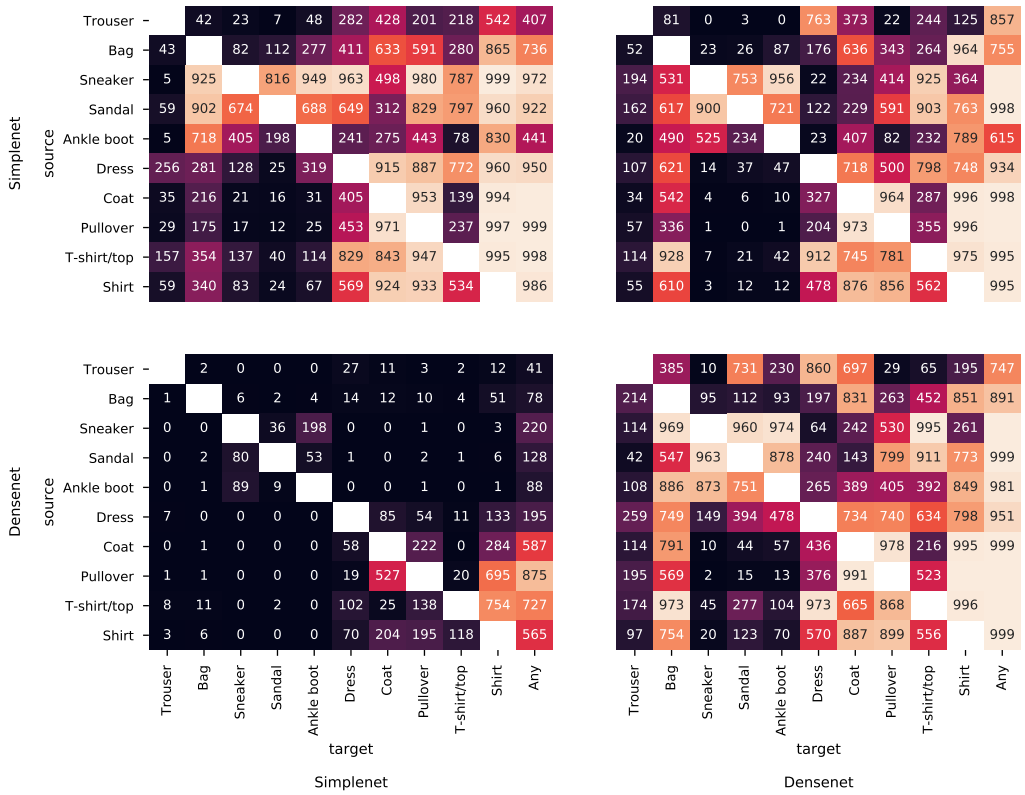

Figure 4.3: Plain surrogate success rate

use $\frac{1}{10}$ of training dataset i.e., 6000 examples. *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 only targeted attack to selected class and non-targeted attack from it. We use preselected median class i.e., *dress*.

Results 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 for plain surrogate (fig 4.3), full surrogate (fig 4.4), reduced surrogate (fig 4.5) and binary surrogate (fig 4.6), we derive following observations.

As expected, surrogate adversarial attack approaches perform inferior to direct white-box methods (see fig 4.2). In detail, full surrogate is the best surrogate approach (fig 4.4), followed by plain and binary surrogate (fig 4.3 and 4.6), with reduced surrogate having the worst performance of all examined scenarios (see fig 4.5).

We observe, that 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, while not being type of footwear or clothing – we think this might be again caused by feature reuse, as from the point of visual similarity, it is close to both of those groups.

SimpleNet is more resilient to attacks than DenseNet (left column of heatmaps scoring lower values than right column). We assume that this might be caused

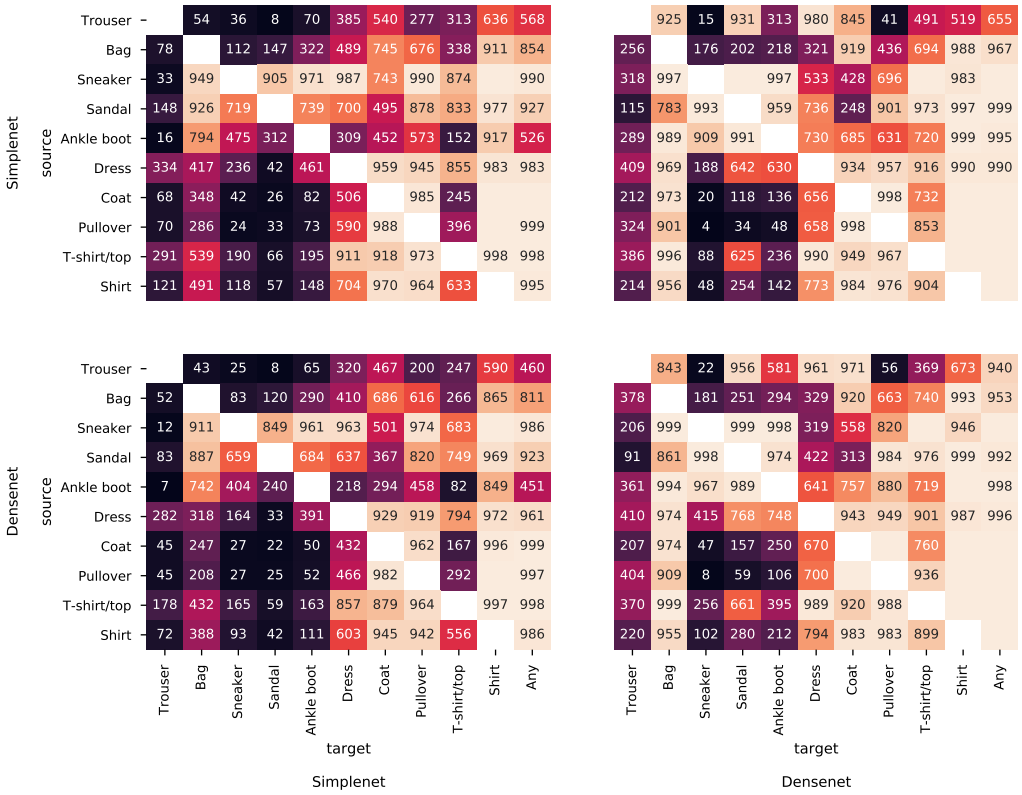

Figure 4.4: Full surrogate success rate

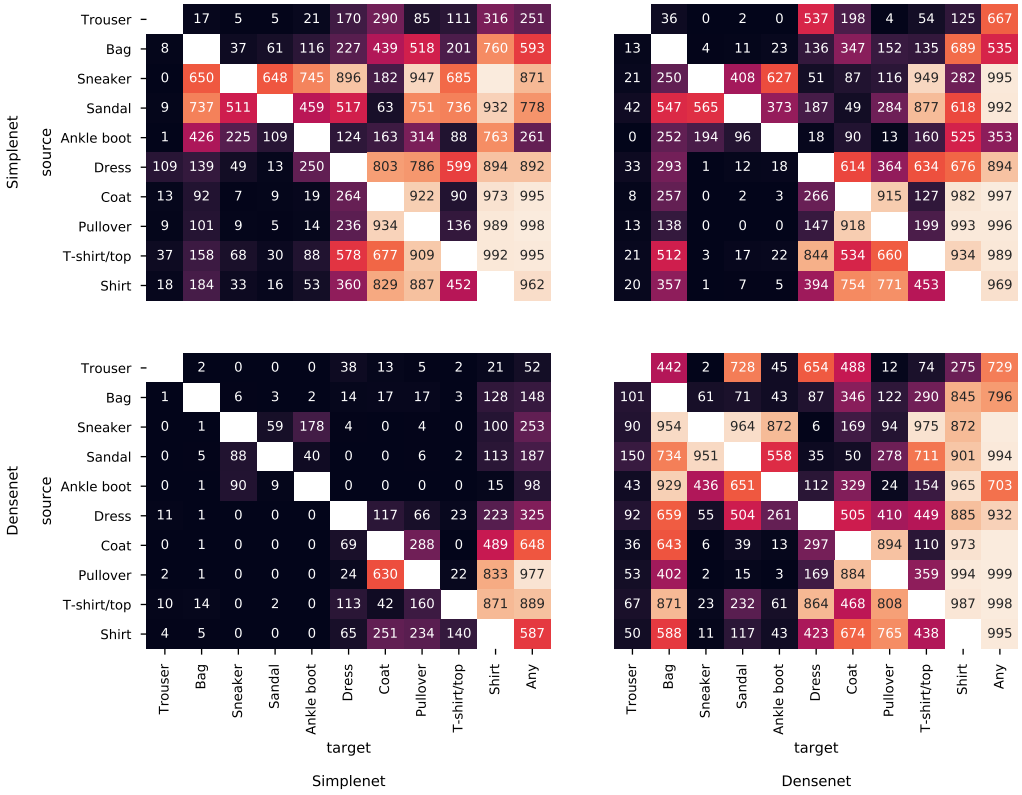

Figure 4.5: Reduced surrogate success rate

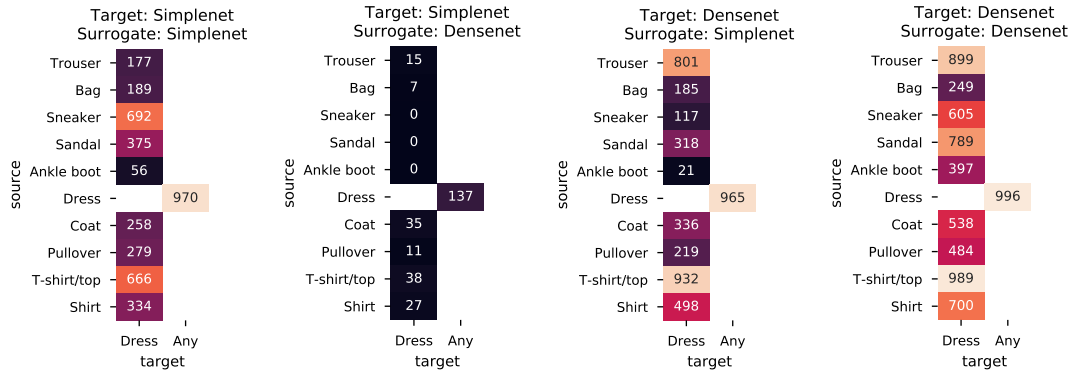

Figure 4.6: Binary surrogate success rate

by extensive feature reuse in case of DenseNet (claimed by Huang et al. [2016]) i.e., one channel of featuremap having significant role in prediction of multiple classes. Consequently, tampering with this shared, class independent feature, influence the prediction of multiple classes, shifting the models response. Simplenet on the other hand may have more class dependent features, forcing the adversary to both counter the evidence for prediction of the source class, while at the same time strenghten the prediction of targeted class.

The interesting phenomenon is the asymmetry in performance of cross architecture attacks, with DenseNet target model attacked by SimpleNet surrogate being far more successful than its counterpart. This again may be caused by feature reuse. We think of it intuitively as of a surjective-only mapping between model features, with multiple SimpleNet features being mapped to one DenseNet feature. As a result, DenseNet is unable to attack specific SimpleNet features. Same architecture attack, on the other hand proves to be successful in both cases, outperforming cross architecture attacks in all cases.

We notice 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. This confirms that both increasing amount of data and increasing knowledge about targeted model improve succes rate of surrogate adversarial attack.

Intristingly enough, the binary attack does not perform as well as we expected, being significantly inferior to full surrogate attack in case of SimpleNet attack, and sligtly inferior in case of DenseNet attack. Our assumption about binary model being by far the best, as it is trained to perfectly fit the boundary between target class and all other classes, proved to be wrong. We explain this by the shift in the task being solved by binary classifier, so that the gradients no longer follow the original full classifier.

TODO reference figures in conclusions??

4.1.3 Black-box Attack

We examine the performance of our proposed solution based on use of genetic algorithms (see section 3.2).

The hyperparameters of the approach are set as follows: the cross-over takes

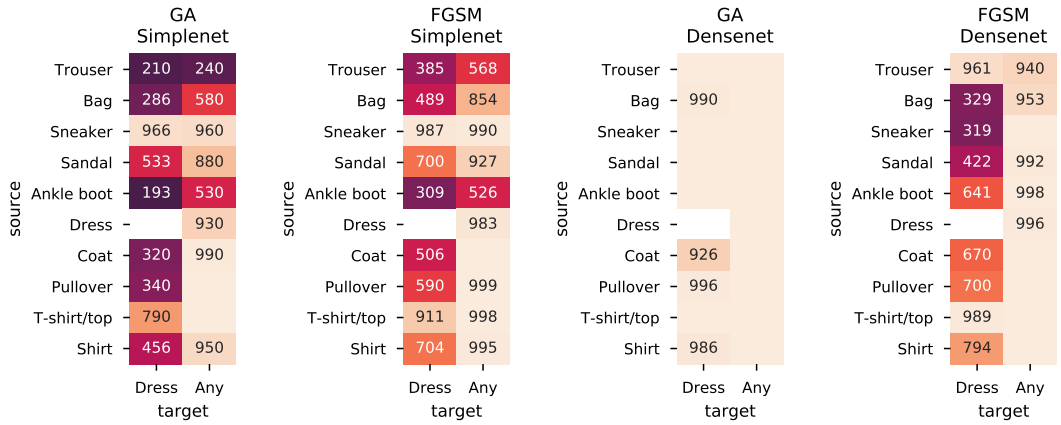

Figure 4.7: Black-box attack success rate (compared to full surrogate)

place with probability $p_{xover} = 0.8$, the biased mutation uses binomial distribution with $n = 26, p = 0.5$ shifted to zero and scaled by factor $\frac{1}{255}$, with $p_{mut} = 0.3$ the probability of individual being mutated and $p_{gene} = 0.15$ the probability of gene mutation. The clipping bound of the perturbation $l2$ norm is set to 0.0005 – 0.95 quantile of $l2$ norm of perturbations in FGSM attacks, for the sake of result comparability. Maximal adversarial example pixel-wise difference is constrained to 0.1 to match the constraints of gradient based method experiments.

Results Regarding the justified increase in computational cost of running black-box attack, particularly genetic algorithm based, we test only two scenarios - median class targeted and non-targeted attack.

We compare the results with the best surrogate approach – full, same architecture, fast gradient sign method surrogate. As showed in fig 4.7, the performance of non-targeted attack is perfect using both approaches. On the other hand targeted attack exhibits significant assymetry regarding the different architectures. The observed assymetry follows the trend of SimpleNet being more resilient to adversarial attacks. SimpleNet black-box attack performance proves to be inferior to that of full surrogate attack on the same architecture while following similar distribution of source – target pairs success rate. On the contrary, black-box attack targeted on DenseNet architecture yield perfect results, superior to those of surrogate attack, clearly showing the unexpected deviation in success rate. We assume this to be caused by genetic algorithm exploration capabilities, which the fast gradient sign method does not offer, sufficient to attack weaker DenseNet model.

The black-box attack yields positive results in terms of number of targeted model evaluations (see fig 4.8). The non-targeted attack on DenseNet architecture reports very low demands, with median complexity under 10 000 calls for each class. SimpleNet, again, proves to be more resilient, with median calls rising up to 30 000 with high variance. It is worth to note, that the distribution of steps needed to be taken to perform successful non-targeted attack against SimpleNet, loosely corresponds to the distribution of success rate of this attack. In case of targeted attack, namely attack targeted on *dress* class, the difference in resiliency between model architectures is reported again. For DenseNet, targeted attack

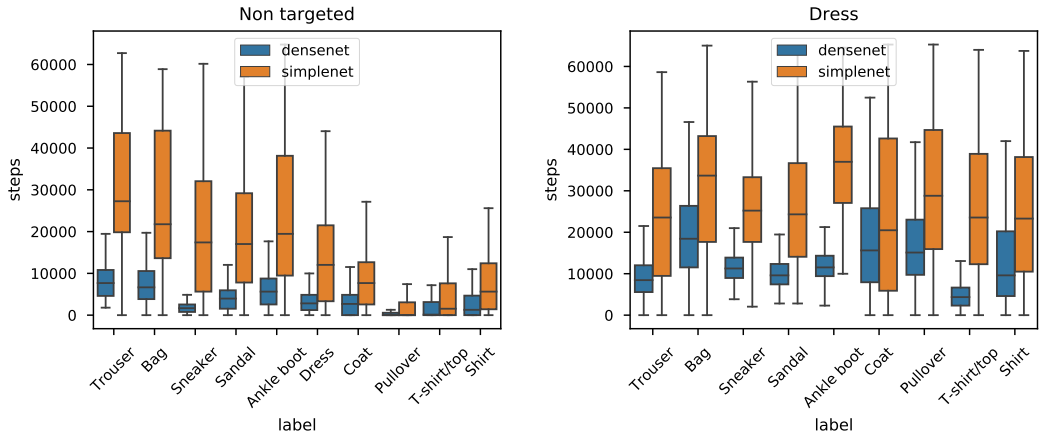

Figure 4.8: Number of black-box model evaluations (compared to full surrogate)

proves to be harder than non-targeted, rising the complexity up to 20 000 median calls for source class *bag*. SimpleNet complexity rised by approximately the same offset, resulting in at most 40 000 median calls.

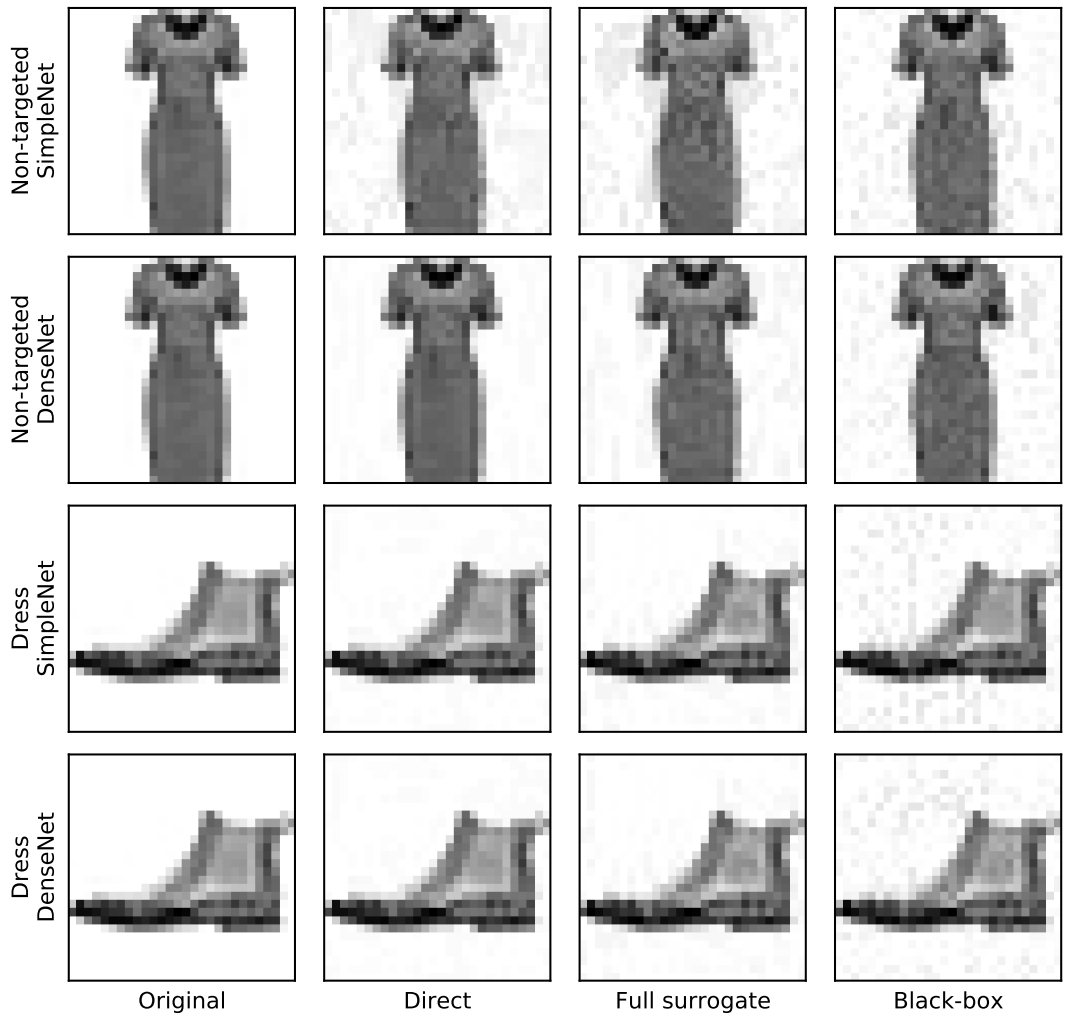

Figure 4.9: Comparison of successful adversarial examples

Conclusion

In our work, we covered the generation of adversarial examples for image classifiers, using fast gradient sign method in white-box and surrogate attack scenario, along with black-box adversarial attack with our proposed genetic algortihm based solution (see section 3.2). We assessed the performance of those methods in terms of success rate and computational complexity.

We carried out a multitude of experiments on publicly available Fashion MNIST dataset (see section 1.1.2), which provides moderately difficult image classification task. With the emphasis on obtaining unbiased and valid observations, we employed model cross-validation and adversarial attack testing on significant amount of examples. The possible discrepancy in attack efficiency between various targeted architectures was assessed using two different architectures, both performing well on the original task (see fig 4.1). Under those test conditions, we got following results.

White-box fast gradient sign method attack yields perfect results (see section 4.1.1), as it exploits the full knowledge of model internal workings, mainly inherent property od deep neural networks – differentiability. While very successful, this type of attack is unfortunately the most impractical for the need to have a full knowledge of the targeted model – an unlikely case for embedded systems or cloud services. On the other hand, white-box fast gradient sign method attack may prove useful during training phase to enhance the adversarial attack resilience of the trained model.

We have carried out four types of surrogate fast gradient sign method attack, all successful to some extent (see section 4.1.2). Non-targeted attack proved to be successful in most scenarios, while targeted attack proved to be substantialy harder and in terms of success rate, heavily source – targeted class dependent. The assumption of binary surrogate attack having significant performance gain over all other surrogate methods proved to be wrong, in fact, the binary surrogate attack even failed to outperform the plain surrogate attack. Simple wide convolutional network (SimpleNet) proved to be more resilient than state of the art deep and narrow DenseNet architecture with residual connections. We account that behaviour to the extensive feature reuse in DenseNet, allowing for confusion of the model with weaker adversary. Surrogate attacks proved to be a promising attack strategy, being moderately successful, while not requiring the full knowledge of the targeted model. Nevertheless, the adversary must still have substantial expert knowledge to build the surrogate model comparable to the targeted one, and a moderate amount of training data to approximate the target model output distribution.

The black-box adversarial attack was carried out using our genetic algorithm based solution. The attack yields satisfactory results in non-targeted scenario. In targeted scenario, compared to full same-architecture surrogate attack, it reports worsened results in case of SimpleNet, on the other hand nearly perfect results in case of DenseNet – even outperforming the surrogate attack performance (see section 4.1.3).

For the future work, we see the potential in extending our solution by informed operator based on surrogate model, hence creating hybrid approach. Such a newly

defined operator is expected to work as a mutation operator within the genetic algorithm, using surrogate model and corresponding gradients obtained with fast gradient sign method. We recognize two possible origins for the surrogate model. Firstly, a *pretrained* surrogate model may be provided during the initialization of the operator – this approach may be suitable for underperforming surrogates to boost their performance with the help of genetic algorithm. Secondly, the surrogate model can be trained on the fly using targeted model predictions obtained from evaluation of objective function. We assume the surrogate model to get to estimate the gradients reliably after few epochs, as the training data i.e., the querries to targeted model are localized in small neighbourhood of the population of genetic algorithm. Using the FGSM mutation operator, the surrogate model may be exploited further as a substitute to targeted model in case of evaluation of objective function. We expect setting the policy of evaluation and training of the surrogate model not to be an easy task, which may even prove to be impractical for having too many hyperparameters sensitive to the correct settings.

We believe that appart from the study of adversarial attacks, study of defenses against those is just as important, if not more. We assume that improvements in dataset augmentation as well as employing methods of adversarial attacks to the process of training may lead to significant improvements in resilience of deep learning models. Appart from that, study of feature reuse and its consequences with respect to adversarial attacks may offer interesting insights.

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.

Charles Darwin. *On the Origin of Species by Means of Natural Selection*. Murray, London, 1859. or the Preservation of Favored Races in the Struggle for Life.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Trans. Evol. Comp*, 6(2):182–197, April 2002. ISSN 1089-778X. doi: 10.1109/4235.996017. URL http://dx.doi.org/10.1109/4235.996017.

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

Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. Cambridge University Press, New York, NY, USA, 2012. ISBN 1107422221, 9781107422223.

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.

Andrew Ilyas, Logan Engstrom, Anish Athalye, and Jessy Lin. Black-box adversarial attacks with limited queries and information. *CoRR*, abs/1804.08598, 2018. URL http://arxiv.org/abs/1804.08598.

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.

N. Narodytska and S. Kasiviswanathan. Simple black-box adversarial attacks on deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1310–1318, July 2017. doi: 10.1109/CVPRW.2017.172.

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.

Nicolas Papernot, Patrick D. McDaniel, Ian J. Goodfellow, Somesh Jha, Z. Berkay Celik, and Ananthram Swami. Practical black-box attacks against deep learning systems using adversarial examples. *CoRR*, abs/1602.02697, 2016. URL http://arxiv.org/abs/1602.02697.

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.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

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.

Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010. ISBN 1848829345, 9781848829343.

Petra Vidnerová and Roman Neruda. Evolutionary generation of adversarial examples for deep and shallow machine learning models. In *Proceedings of the The 3rd Multidisciplinary International Social Networks Conference on Social-Informatics 2016, Data Science 2016*, MISNC, SI, DS 2016, pages 43:1–43:7, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4129-5. doi: 10.1145/2955129.2955178. URL http://doi.acm.org/10.1145/2955129.2955178.

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.

Xiaoyong Yuan, Pan He, Qile Zhu, Rajendra Rana Bhat, and Xiaolin Li. Adversarial examples: Attacks and defenses for deep learning. *CoRR*, abs/1712.07107, 2017. URL http://arxiv.org/abs/1712.07107.

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

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012, 2017. URL http://arxiv.org/abs/1707.07012.

Štěpán Procházka. Evolutionary generated adversarial examples, 2018. URL https://github.com/proste/evgena. Last visited 2018-07-20.

List of Figures

List of Algorithms