fhwedel
UNIVERSITY OF APPLIED SCIENCES

Department of Computer Science

Master Thesis

Adversarial Attacks and Defenses for Image-Based Recommendation Systems using Deep Neural Networks

Submitted on:

August 26, 2020

Submitted by:
Philipp Normann
E-mail: philipp.normann@otto.de

Advisor (FH Wedel):
Prof. Dr. Gerd Beuster
Fachhochschule Wedel
Feldstraße 143
22880 Wedel
E-mail: gb@fh-wedel.de

Advisor (OTTO):
Christian Stamm
Otto GmbH & Co KG
Werner-Otto-Straße 1-7
22761 Hamburg
E-mail: christian.stamm@otto.de

Abstract

Today recommendation systems (RSs) are an established part of modern web-services and are deployed by numerous companies. Owing to the success of deep neural networks (DNNs) in other domains, the industry has started implementing RSs using DNNs. However, DNNs were shown to be vulnerable to targeted adversarial attacks in recent studies. While there have been several studies on the subject of adversarial attacks against collaborative filtering (CF) based RSs only few studies focusing on content-based RSs have been published. In this thesis, we showed that a visual content-based RSs using DNNs is vulnerable to targeted adversarial attacks using state-of-the-art white-box attacks. In the next step, we tested different defense mechanisms utilizing adversarial training (AT) and were able to show that AT had a significant positive impact on the robustness of our trained models against our performed attacks.

Contents

List of Acronyms

API	application programming intefrace
ANN	artificial neural network
AT	adversarial training
BCE	binary cross-entropy
BIM	basic iterative method
CAT	curriculum adversarial training
CBF	content-based filtering
CE	cross-entropy
CF	collaborative filtering
CNN	convolutional neural network
CW	Carlini & Wagner
DAG	directed acyclic graph
DDoS	distributed denial-of-service
DL	deep learning
DNN	deep neural network
DSS	decision support system
FGSM	fast gradient sign method
HNSW	hierarchical navigable small world
IDS	intrusion detection system
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
MF	matrix factorization
ML	machine learning
MNIST	Modified National Institute of Standards and Technology
MSE	mean squared error
NCF	neural collaborative filtering
NLP	natural language processing
OCR	optical character recognition
PGD	projected gradient descent
RNN	recurrent neural network
RS	recommendation system
ReLU	rectified linear unit
SGD	stochastic gradient descent
k-NN	k-nearest neighbors
t-SNE	t-distributed stochastic neighbor embedding

1 Introduction

Recommendation systems (RSs) are a special kind of decision support system (DSS) that can help users quickly find what they desire or guide them towards new content they might find appealing. These kinds of systems have reached increasing industrial adoption in recent years. Today, numerous companies, ranging from e-commerce marketplaces [1] [2] to music [3] and video [4] streaming services, as well as social networks [5] and news aggregators [6], successfully deploy recommendation systems (RSs) to improve the user experience of their services and subsequently gain a measurable competitive advantage, as demonstrated by P.-Y. Chen et al. (2004).

However, while these systems affect a lot of our daily decisions, they are also exposed to external threats from malicious actors, which try to exploit these RSs to their advantage, through targeted manipulation of the data used for RS training and application. The goals of attackers can range from promoting their products to manipulating public opinion on a specific topic. Depending on the application area of the RS, a successful compromise can have far-reaching consequences. Therefore, a better understanding of potential attacks and defenses is essential to develop reliable and robust systems, which we can trust.

Adversarial examples are inputs to a machine-learning model that are intended to force the model to make incorrect predictions and were first formally described by Dalvi et al. (2004) when researchers studied the techniques used by spammers to circumvent spam filters. In particular, deep neural networks but also many other categories of machine learning models are highly vulnerable to attacks based on small modifications of the input to the model at prediction time, as first demonstrated by Szegedy et al. (2013). Since then, the topic of adversarial examples has developed into a fast-growing research field (Carlini, 2019).

Researchers have also successfully performed adversarial attacks against real-world RSs, such as YouTube, Google Search, Amazon, and Yelp in experiments (Xing et al., 2013; Yang et al., 2017). Large companies like Sony, Amazon, and eBay have also publicly reported that they have suffered from such attacks in practice (Lam & Riedl, 2004).

Most previous work on the potential attacks against and defenses for RSs has focused on traditional collaborative filtering methods, particularly matrix factorization (MF) algorithms. Various attacks, such as the injection of handcrafted fake profiles, also

[1] Otto GmbH & Co KG https://www.otto.de
[2] Amazon.com Inc. https://www.amazon.com
[3] Spotify Technology S.A. https://www.spotify.com
[4] Netflix Inc. https://www.netflix.com
[5] Facebook Inc. https://www.facebook.com
[6] Google Inc. https://news.google.com

called shilling attacks (Chirita et al., 2005) or, more recently, automatically generated fake user profiles (Christakopoulou & Banerjee, 2019), have been studied. However, only very little research, with a focus on content-based RSs, especially those utilizing deep learning techniques and their exploitability by adversarial examples, has been published. While two previous works have already explored the general vulnerability of content-based RSs using convolutional neural networks (CNNs) to untargeted attacks (Tang et al., 2019) and targeted category misclassification attacks (Di Noia et al., 2020), no previous study has explored the feasibility of targeting specific recommendation slots and items using adversarial attacks.

This thesis closes this research gap by developing targeted attacks and defenses using standard techniques from the field of adversarial examples to a content-based RS. Our goal is to investigate the vulnerability of a deep learning-based RSs to our proposed attacks and to evaluate the effectiveness of published defense techniques.

1.1 Structure

The chapters of this thesis are structured in the following way:

- **Chapter 2**: Provides the relevant theoretical background for the context of deep learning, specifically adversarial learning and recommendation systems. This background information includes topics such as supervised learning, image classification, optimization of parametric models, artificial neural networks (ANNs), backpropagation, popular recommendation frameworks, and adversarial attacks.
- **Chapter 3**: Performs a systematic analysis of the adversarial goals and capabilities concerning content-based RSs.
- **Chapter 4**: Provides an overview of the DeepFashion dataset used for training the image-based recommendation system.
- **Chapter 5**: Defines the type of image-based recommendation system that we used as the foundation for our subsequent robustness studies.
- **Chapter 6**: Presents our proposed targeted item-to-item attacks and discuss our experimental results.
- **Chapter 7**: Evaluates two published defense techniques against adversarial inputs for our recommendation system.
- **Chapter 8**: Presents a conclusion of our experiments and discuss the current state of research regarding adversarial attacks on RSs using deep neural networks (DNNs).

2 Background

This chapter provides an overview of the theoretical prerequisites on machine learning (ML), recommendation system (RS) and the field of adversarial examples that were required for solving the main task of this thesis.

2.1 Supervised Learning

The majority of problems from the field of ML require a parameterized algorithm f_θ to perform a mapping $f_\theta : X \to Y$, where X is the input space and Y the output space. For example, in visual recognition, X might be the space of images, while Y could be the interval $[0, 1]$ describing the probability of a dog appearing in it. Since it is often not trivial to manually specify this function f by traditional means, the supervised learning paradigm offers an alternate approach, which takes advantage of the fact that in many cases, it is relatively easy to acquire a dataset containing examples of the desired mapping. Using these observations, it is possible to train a parametric model, approximating function f iteratively. After successful training, this approximated function can then predict values of Y for unobserved inputs of X (Karpathy, 2016).

In order to find an acceptable approximation, we need an objective function for measuring the disagreement between a model's prediction $\hat{y} = f_\theta(x)$ and the ground truth y, i.e., $\mathcal{L}(\hat{y}, y) \to \mathbb{R}$. Such a function is usually known as the *loss* or *error* function and is the primary quality measure used while training a model. A widely used loss function for regression problems is the mean squared error (MSE):

$$\mathcal{L}_{MSE}(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2 \,, \tag{2.1}$$

while in classification tasks, the cross-entropy (CE) loss function is more commonly used and measures the disagreement between two probability distributions, y and $\hat{y}$:

$$\mathcal{L}_{CE}(\hat{y}, y) = -\sum_{k=1}^{K} (y_k \log \hat{y}_k) = -\log \hat{y}_{y=1} \,. \tag{2.2}$$

Minimizing the task-specific loss function becomes the main objective of the supervised learning task. Usually, a subset of the collected observation is held out for evaluating the model's performance on unseen data. The achieved predictive quality on this test set after training is an indicator of the generalizability of the approximated model f_θ.

2.2 Image Classification

A particularly common supervised learning task is image classification, also known as object recognition. For these types of problems, inputs have the form of 2D pixel intensity matrices in the case of grayscale images and 3D matrices containing separate channel intensities in the case of color images. The channels usually account for the red, green, and blue components of the RGB color space. The outputs are vectors of predicted class probabilities for the given input image, and the goal of the model is to achieve the highest possible accuracy on the test set of unseen images.

Over the years, the research community has established multiple benchmark datasets and competitions for this domain. Beginning with the relatively simple task of handwritten digit recognition, also known as the Modified National Institute of Standards and Technology (MNIST) dataset (LeCun & Cortes, 2010), as seen in Figure 2.1, and evolving into more realistic and complex scenarios like the ImageNet challenge (Russakovsky et al., 2015), also known as the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), as seen in Figure 2.2, where the dataset contains over 14 million images from over 20,000 classes.

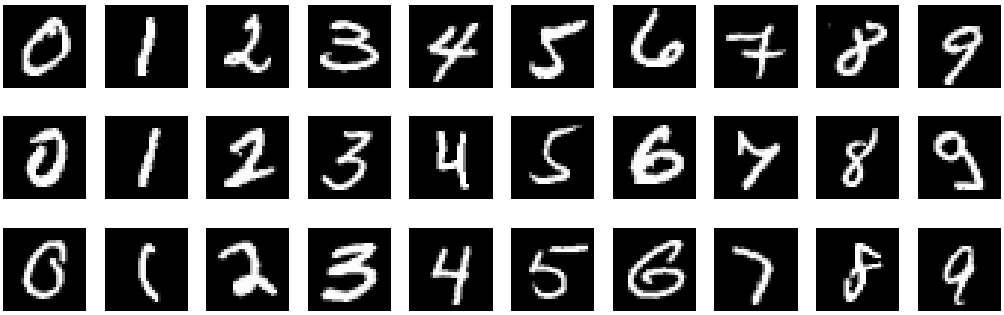


Figure 2.1: Randomly sampled images of each class from the MNIST digit datset.

(a) snail (b) bridge (c) gazania (d) meerkat (e) bok choy

(f) lobster (g) basenji (h) gorilla (i) shark (j) piano

Figure 2.2: Randomly sampled images from the ImageNet challenge.

2.3 Optimization

In Section 2.1, we introduced the concept of a loss function $\mathcal{L}$, which measures the performance of a prediction. In order to optimize a parameterized model f_θ, we also need an algorithm that can derive a parameter update aimed at minimizing this loss function.

The problem of general optimization is given a function $f : X \to Y$ to find values $\hat{x}$ that minimizes f, i.e., $\hat{x} = \arg\min f(x)$. To optimize a differentiable function f, one commonly used method is called *gradient descent*. Starting at a randomly initialized value for $\hat{x}$, gradients $\frac{\partial f}{\partial \hat{x}}$ are calculated to iteratively adjust the parameters $\hat{x}$ towards the negative direction of the gradients. The gradient is a vector of partial derivatives, representing the slope of the function along each dimension of the parameter space, i.e. given $f(x_1, ..., x_n)$ the gradient is defined as $\nabla f = \left(\frac{\partial f}{\partial x_1}, ..., \frac{\partial f}{\partial x_n}\right)^T$. In cases where analytically solving an optimization problem is computationally infeasible, *gradient descent* can be used to iteratively minimize a differentiable function as seen in Figure 2.3. In the case of optimizing parameterized models using a loss function $\mathcal{L}$, as presented in Section 2.1, we calculate the gradients of our loss function in respect to the parameters of the model $\nabla_\theta \mathcal{L}(f_\theta(x_i), y_i)$ and use gradient descent to iterativly optimize our model.

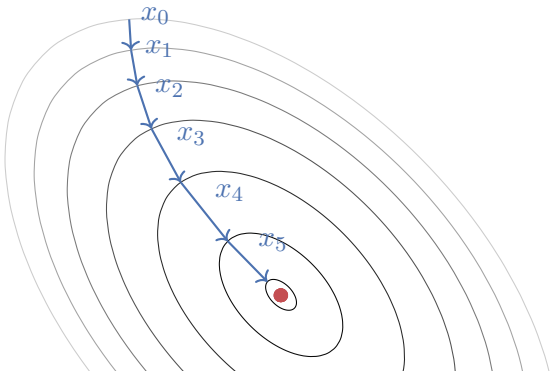


Figure 2.3: Visualization of the *gradient descent* algorithm where steps are taken following the direction of the slope (negative gradient) at each iteration. Here x_0 to x_5 represent data points on the contour plot of a loss function whose minimum is at the red dot in the center.

In practice, datasets can be very extensive, containing millions of training examples. This circumstance makes the exact calculation of gradients across the whole training data, computationally infeasible. Therefore, the gradients are usually estimated using small batches of examples, resulting in a larger amount of approximate parameter updates, instead of fewer exact ones. This method speeds up training and has proven to work well in most practical use cases (Shamir & Zhang, 2013). Algorithm 1 depicts the individual steps of this procedure. The described algorithm is called stochastic gradient descent (SGD) and is the basis for most optimizers used for training deep neural networks (DNNs).

Algorithm 1 Stochastic Gradient descent (SGD)

Require: Training data $\mathcal{D}$; Learning rate η;
1: $\theta \leftarrow$ random parameter initialization
2: **repeat**
3: $(x_i, y_i) \leftarrow$ sample batch from training data $\mathcal{D}$
4: $\theta \leftarrow \theta - \eta \sum_i \nabla_\theta \mathcal{L}(f_\theta(x_i), y_i)$
5: **until** convergence

The step size, also known as the learning rate, is a critical parameter for the algorithm. If it is too large, the SGD will overshoot local minima, since the slope of the function is constantly changing. Vice versa, if it is too small, a minimum will be found reliably, but the convergence is slow since recalculating the gradients for each step is time-consuming. More advanced variants of SGD like AdaGrad (Duchi et al., 2011), Adam (Kingma & Ba, 2014) or RMSProp (G. Hinton, Srivastava, et al., 2012) are used in practice to speed up convergence. They achieve this by taking into account previous gradients and momentum while computing the updates for an optimization step (Algorithm 1, Line 4).

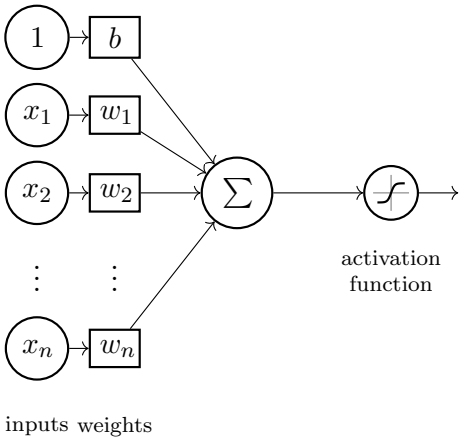
2.4 Artificial Neural Networks

The previous sections described how a parametric model f_θ, consisting of differentiable functions, can be used to transform the inputs x_i to predicted outputs $\hat{y}_i = f_\theta(x_i)$ and how the parameters of such a model can be iteratively optimized using the SGD algorithm. This section further details the definition of this function f_θ for traditional feedforward networks and convolutional neural networks (CNNs), a particular type of artificial neural network (ANN), which is especially useful for image processing tasks.

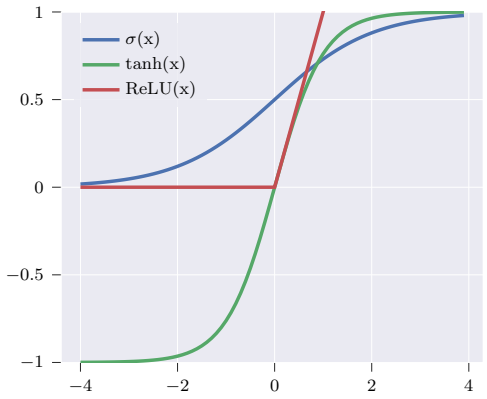
2.4.1 Feedforward Neural Network

A feedforward network is characterized by its unidirectional dataflow, meaning that the graph representing the network contains no cyclic connections. Data enters at the inputs and is transformed through the network layers by repeating matrix multiplications and element-wise non-linearities until reaching the output. Feedforward neural networks usually consist of multiple layers, as seen in Figure 2.5. These layers are made up of computational units that fully connect to their next layer's units. In the context of ANNs the output of a single hidden neuron or computational unit is referred to as its activation. These activations, as seen in Figure 2.4, are calculated using a biased weighted sum of their inputs x_i and apply a nonlinear activation function φ to the result, i.e.

$$\hat{y} = f_w(x) = \varphi\left(\sum_{i=0}^{n} x_i w_i + b_i\right) \tag{2.3}$$



(a) Visualization of an artifical hidden neuron, computing the result of Eqution 2.3.



(b) Plots of the three most common nonlinear activation functions used in ANNs.

Figure 2.4: The anatomy of an artificial hidden neuron

Hornik et al., 1989 first demonstrated that feedforward networks with a sufficiently large amount of hidden units can be considered universal function approximators. Since then, researchers have applied these types of ANNs successfully to a wide range of real-world tasks, such as optical character recognition (OCR) (LeCun et al., 1989) and speech recognition (Morgan & Bourlard, 1990). Invented in the 1980s, they are the oldest types of neural networks and have been the basis for more complex architectures such as CNNs and recurrent neural networks (RNNs).

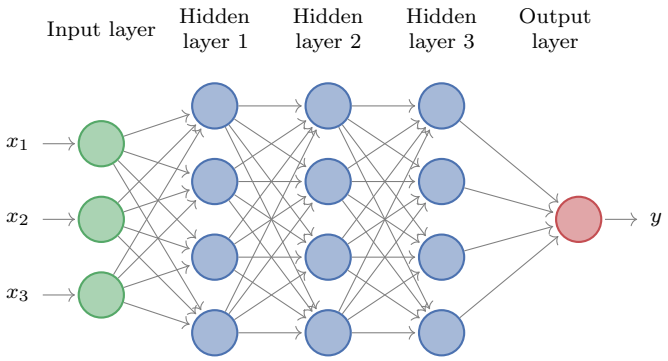


Figure 2.5: Graph for a multilayer ANN, containing three hidden layers. All nodes of each layer are fully connected to the next layer.

2.4.2 Convolutional Neural Networks

Convolutional neural networks (CNNs), also known as ConvNets, first introduced by LeCun et al., 1989, are a specialized kind of neural network for processing data with a known grid-like spatial topology (e.g., images, videos, sound spectrograms, character sequences or 3D voxel data). In each of these cases, an input example x is a multi-dimensional array (tensor). E.g. a 256×256 color image is a $256 \times 256 \times 3$ tensor (for 3

color channels red, green, blue). In many of these cases, the input dimensionality is high (e.g., the image above will have approximately 200,000 numbers), and it is therefore infeasible (in both the number of parameters and processing time) to use fully-connected layers as we saw in Section 2.4.1. In these cases, we prefer to design neural network architectures that take advantage of the spatial input topology by using specific local connectivities and reasonable parameter sharing schemes.

The convolutional operation, the core computational building block that gave CNNs their name, provides a solution to this problem. A convolution is a specialized kind of linear operation, and CNNs are neural networks that use convolution in place of general matrix multiplication in at least one of their layers (I. Goodfellow et al., 2016).

ConvNets are a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures – convolutional, pooling, and fully connected, as already seen in traditional feedforward networks from Section 2.4.1. We will stack these layer types to form a full ConvNet architecture. In the following sections, we will take a closer look at these unique layer types and their modes of operation.

Convolutional Layer

In the general case, a discrete convolution operation implements the function:

$$s(i) = \sum_{a=-\inf}^{\inf} x(a)k(i-a) \tag{2.4}$$

where x is the input and k a weighting function known as a kernel. In the case of image processing CNNs, the input I, as well as the kernels K, also known as filters, are typically two-dimensional so that we can write out the convolution function with both axes (i, j) as (Stanford, 2020):

$$S(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{2.5}$$

The convolutional layer's learnable parameters consist of a set of filters. During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume, we will produce a 2-dimensional activation map that gives the responses of that filter at every spatial position. We can visualize this procedure, as seen in Figure 2.6.

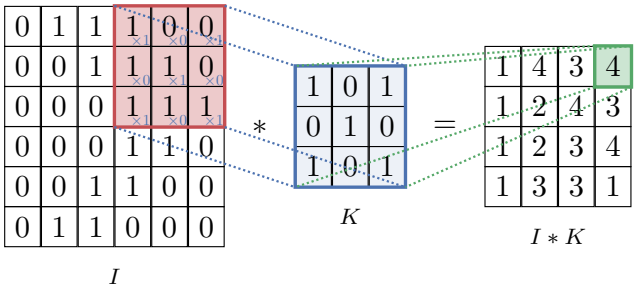


Figure 2.6: A diagram expressing a two-dimensional convolutional operator as an operation of sliding the kernel matrix K across the target image I and recording elementwise products.

Pooling Layer

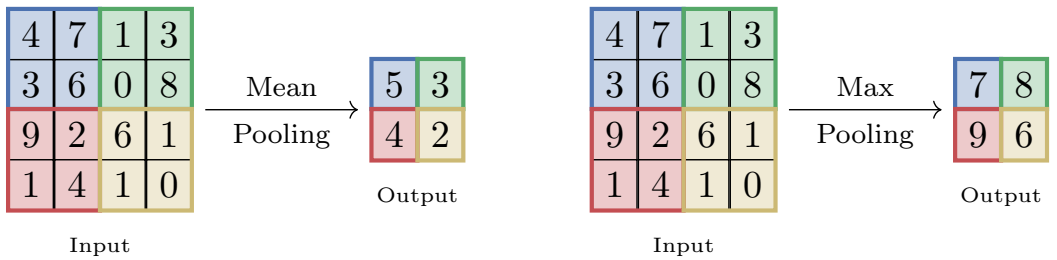
To further control the computational complexity and achieve invariance to small translations of the input, it is common practice to use pooling layers to decrease the representation with a fixed downsampling transformation (i.e., without any learnable parameters). In particular, the pooling layers operate on each activation map independently and downsample them spatially (Karpathy, 2016). Two commonly used pooling operations are mean-pooling and max-pooling. The mean-pooling operation is defined as:

$$\frac{1}{|u||v|} \sum_{i \in u} \sum_{j \in v} I(i,j) \tag{2.6}$$

while the max-pooling operation is defined as:

$$\max_{i \in u, j \in v} I(i,j). \tag{2.7}$$

In both cases, u, v are vectors of kernel indices and a visual representation of both pooling operations is provided in Figure 2.7.



(a) Visualization of the *mean pooling* operation. Color coded patches are combined via arithmetic average.

(b) Visualization of the *max pooling* operation. Color coded patches are downsampled by taking the maximum value found in the patch.

Figure 2.7: Common pooling operations used in CNNs.

ConvNet Architectures

We have seen that CNNs are commonly made up of only three layer types: convolutional (`CONV`), pooling (`POOL`), and fully-connected (`FC`). We will also explicitly write the activation function as a layer (e.g., `RELU`), which applies an elementwise non-linearity. In this section, we discuss how these layers are commonly stacked together to form entire ConvNets and which architectures have achieved state-of-the-art performance on the ImageNet challenge.

The most common form of a ConvNet architecture stacks a few `CONV-RELU` layers, follows them with `POOL` layers and repeats this pattern until the image has been merged spatially to a small size. At some point, it is common to transition to fully-connected layers. The last fully-connected layer holds the output, such as the class scores. In other words, the most common ConvNet architecture follows the pattern:

```
INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC
```

where the `*` indicates repetition, and the `POOL?` indicates an optional pooling layer. Additionally, `N >= 0` (and usually `N <= 3`), `M >= 0`, `K >= 0` (and usually `K < 3`).

CNNs have been successfully applied to image classification tasks. They were first used by LeCun et al., 1989 to recognize digits from the MNIST dataset and were then widely popularized when Krizhevsky et al., 2012 won the ILSVRC 2012 by a large margin with their CNN architecture known as AlexNet. Subsequent editions of the challenge were also won by CNNs like GoogLeNet (Szegedy et al., 2015) or ResNet (K. He et al., 2016).

The runner-up in ILSVRC 2014 was the network from Simonyan and Zisserman, 2014 that became known as VGGNet. Its main contribution was in showing that the depth of the network is critical for good performance. Their final best network, known as VGG-16, visualized in 2.8, contains 16 `CONV/FC` layers and features an extremely homogeneous architecture that only performs 3×3 convolutions and 2×2 pooling from the beginning to the end (Stanford, 2020).

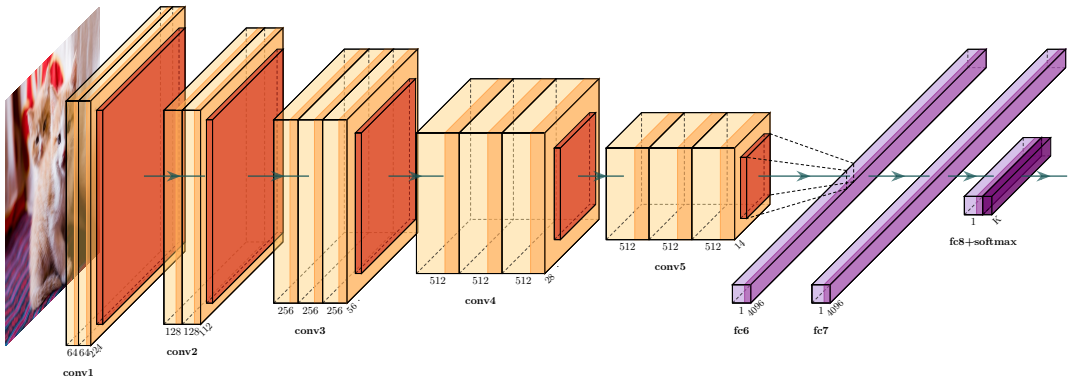


Figure 2.8: Diagram of the CNN architecture of VGG-16

2.5 Backpropagation

The generic approach of applying SGD to compositional models, such as ANNs, is called backpropagation. Due to the composite form, we can calculate the gradients using the chain rule for differentiation (Hastie et al., 2009, p. 395). For a better intuition as to why this is possible, we can look at the functions that the ANN in Figure 2.9 consists of:

$$h_i = \varphi\left(\sum_{j=0}^{n} x_j w_{h_i}^{x_j}\right) , \tag{2.8}$$

$$\hat{y} = \sum_{i=0}^{m} h_i w_{\hat{y}}^{h_i} , \tag{2.9}$$

where φ is a differentiable activation function, n represents the number of inputs, m equals the number of hidden neurons, x_j refers to the inputs and w_x^y are the weights in neuron x for input y.

The partial derivative of the loss function $\mathcal{L}$, with respect to a weight $w_{h_i}^{x_j}$ in a hidden neuron h_i for input x_j, results from applying the chain rule

$$\frac{\partial \mathcal{L}}{\partial w_{h_i}^{x_j}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial \mathrm{h}_i} \frac{\partial \mathrm{h}_i}{\partial w_{h_i}^{x_j}} . \tag{2.10}$$

The resulting value indicates, in which direction the respective weight $w_{h_i}^{x_j}$ has to be adjusted by an infinitesimally small amount, in order to decrease the loss $\mathcal{L}$ (Rojas, 2013, p. 169). For simplicity reasons, the bias of each neuron is omitted.

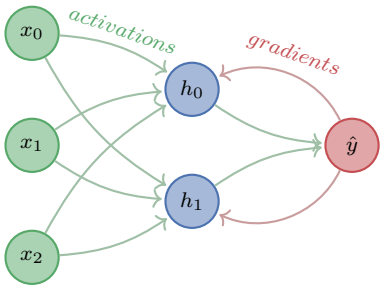


Figure 2.9: Backpropagation in a single hidden layer ANN. After a set of inputs is processed in the forward pass (green), the final error is determined and the gradients (contribution of neurons to error) are calculated recursively using the chain rule in the backward pass (red).

2.5.1 Numerical Example

In order to further illustrate the process of backpropagation, a complete forward and partial backward pass of the ANN in Figure 2.9 is conducted.

Using a rectified linear unit (ReLU) as an activation function:

$$\varphi(x) = max(0, x) \tag{2.11}$$

Note. ReLU is technically not differentiable at $x = 0$, but is still widely used, due to its empirical performance. It is common practice to set the derivative at this discontinuity to zero or one (Nair & Hinton, 2010, p. 4).

Given inputs x_i and a label y:

$$x_0 = 0 \qquad x_1 = 1 \qquad x_2 = 0 \qquad y = 1 \tag{2.12}$$

Weights for all hidden and output neurons:

$$\begin{aligned} w_{h_0}^{x_0} &= 0.0 & w_{h_0}^{x_1} &= 0.1 & w_{h_0}^{x_2} &= 0.2 \\ w_{h_1}^{x_0} &= 0.2 & w_{h_1}^{x_1} &= 0.4 & w_{h_1}^{x_2} &= 0.6 \\ w_{\hat{y}}^{h_0} &= 0.3 & w_{\hat{y}}^{h_1} &= 0.2 \end{aligned} \tag{2.13}$$

Activation of the first hidden neuron h_0:

$$\begin{aligned} h_0 &= \varphi(x_0 w_{h_0}^{x_0} + x_1 w_{h_0}^{x_1} + x_2 w_{h_0}^{x_2}) \\ &= \max(0,\, 0 \cdot 0.0 + 1 \cdot 0.1 + 0 \cdot 0.2) \\ &= 0.1 \end{aligned} \tag{2.14}$$

Activation of the second hidden neuron h_1:

$$\begin{aligned} h_1 &= \varphi(x_0 w_{h_1}^{x_0} + x_1 w_{h_1}^{x_1} + x_2 w_{h_1}^{x_2}) \\ &= \max(0,\, 0 \cdot 0.2 + 1 \cdot 0.4 + 0 \cdot 0.6) \\ &= 0.4 \end{aligned} \tag{2.15}$$

Activation of the output neuron $\hat{y}$:

$$\begin{aligned} \hat{y} &= h_0 w_{\hat{y}}^{h_0} + h_1 w_{\hat{y}}^{h_1} \\ &= 0.1 \cdot 0.3 + 0.4 \cdot 0.2 \\ &= 0.11 \end{aligned} \tag{2.16}$$

The squared error loss $\mathcal{L}$ is calculated with respect to expected label y:

$$\mathcal{L} = (\hat{y} - y)^2 = (0.11 - 1)^2 = 0.7921 \tag{2.17}$$

Since $\mathcal{L}$ is a composite function of $\hat{y}$ (which in turn is again a composite function of h_1 and h_0), the chain rule of differentiation has to be applied, in order to compute the gradients of $\frac{\partial \mathcal{L}}{\partial w_{\hat{y}}^{h_0}}$ and $\frac{\partial \mathcal{L}}{\partial w_{\hat{y}}^{h_1}}$:

$$\frac{\partial \mathcal{L}}{\partial w_{\hat{y}}^{h_0}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{\hat{y}}^{h_0}} = 2(\hat{y} - y) \cdot h_0 = -0.178 \tag{2.18}$$

$$\frac{\partial \mathcal{L}}{\partial w_{\hat{y}}^{h_1}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{\hat{y}}^{h_1}} = 2(\hat{y} - y) \cdot h_1 = -0.712 \tag{2.19}$$

The resulting gradients indicate the sensitivity of $\mathcal{L}$ for $w_{\hat{y}}^{h_0}$ and $w_{\hat{y}}^{h_1}$. In other words how much $\mathcal{L}$ would change if $w_{\hat{y}}^{h_0}$ or $w_{\hat{y}}^{h_1}$ were adjusted. For a complete backward pass, the gradients for all weights of h_1 and h_0 would be calculated, by further chaining the partial derivatives. Assuming the optimizer algorithm is SGD, the results would be multiplied by an infinitesimally small *learning rate* and subtracted from the respective weights during an update step, resulting in a slightly lower value of $\mathcal{L}$.

2.5.2 Computational Graph

It is often more intuitive to think of the functions that make up an ANN, as a directed acyclic graph (DAG), instead of a linear sequence of operations. Vectors flow along the edges of this graph and nodes perform differentiable transformation on these vectors. Most implementations of backpropagation use some kind of graph representation that keeps track of all nodes and operations inside this graph.

The graph and nodes both implement a `forward()` and `backward()` operation. During the `forward()` pass, the graph calls all nodes' `forward()` operations in the correct topological order, each time supplying the outputs of the last node as inputs to the next.

The graph's `backward()` implementation operates in the reverse order, starting with the loss and recursively chaining (multiplying) the nodes' gradients together until the end of the graph is reached. *PyTorch*—the deep learning framework used in the context of this thesis—also makes use of such graph representation (Paszke et al., 2019).

2.6 Recommendation Systems

Recommendation systems (RSs) are decision support systems (DSSs) which analyze patterns of user behavior to provide them with personalized content recommendations that suit their preference. The main reason for online services to deploy recommendation systems, is to handle the phenomenon of information overload, which many users face in a world of evergrowing options and available content (Bobadilla et al., 2013).

More formally, Adomavicius and Tuzhilin, 2005 define the recommendation problem as follows: Let C be the set of all users and let S be the set of all possible items that can

be recommended. Let u be a utility function that measures the usefulness of items to the user c, i.e., $u : C \times S \rightarrow R$, where R is an ordered set. Then, for each user $c \in C$, we want to choose such items $s' \in S$ that maximizes the user's utility. More formally:

$$\forall_{c \in C}, \quad s'_c = \arg\max_{s \in S} u(c, s). \tag{2.20}$$

Estimating this utility function, which optimizes a particular performance criterion, is the central part of any RS. The utility function for items that a user has not interacted with can be estimated in many different ways using methods from machine learning, approximation theory, and various heuristics. RSs are usually classified according to their approach to utility estimation (Adomavicius & Tuzhilin, 2005), as seen in Figure 2.10.

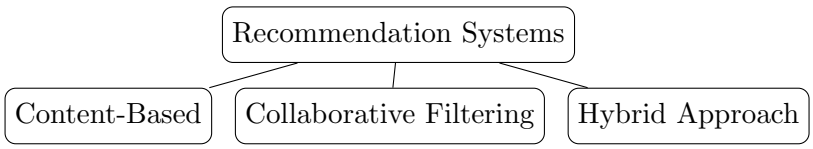


Figure 2.10: Typical categorization for RSs.

In the following sections, we will review the two main approaches of collaborative filtering (CF) and content-based filtering (CBF), explaining the basic concepts and referencing current state-of-the-art models for both approaches.

2.6.1 Collaborative Filtering

Collaborative Filtering (CF) is based on how humans have made decisions throughout history: Besides on our own experiences, we also base our decisions on the experiences and knowledge that reach each of us from a relatively large group of acquaintances (Bobadilla et al., 2013). Therefore CF approaches make recommendations to each user based on the either explicit (e.g., ratings) or implicit (e.g., views or purchases) feedback of users who have the most in common with them, as visualized in Figure 2.11.

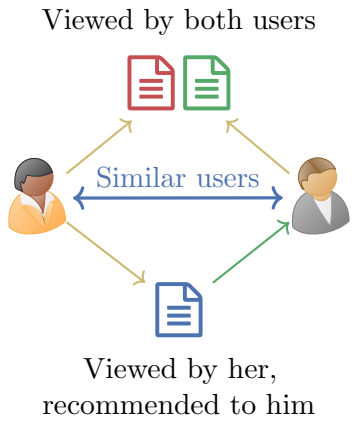


Figure 2.11: Visualization of the collaborative filtering technique

Among the various collaborative filtering techniques, matrix factorization (MF) (X. He et al., 2016) is the most popular one. This approach projects users and items into a shared latent space, using a vector of latent features to represent a user or an item. After that, users' interaction on an item is modeled as the inner product of their latent vectors. Popularized by the Netflix Prize, MF has become the defacto standard approach to latent factor model-based recommendation (X. He et al., 2017). Recently an increasing number of researchers have shown interest in employing DNNs for CF RSs. Prominent examples from this direction of research include the Wide & Deep model from Google (Cheng et al., 2016), the neural collaborative filtering (NCF) approach published by X. He et al., 2017 and the DeepFM model by Guo et al., 2017.

2.6.2 Content-Based Filtering

Content-based filtering (CBF) generates recommendations using the content from items intended for the recommendation; therefore, specific content needs to be analyzed, like text, images, or sound. A similarity measure between objects can be established from this analysis serving as the basis for recommending items similar to items that a user has bought, visited, heard, viewed, or ranked positively (Bobadilla et al., 2013). This general approach is visualized in Figure 2.12.

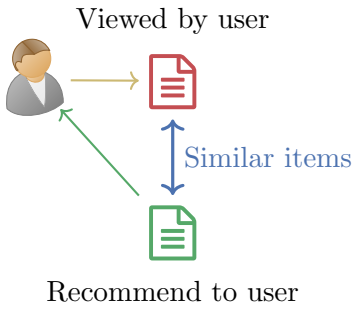


Figure 2.12: Visualization of the content-based filtering technique

For a CBF system to operate, attributes of the items intended for recommendation must be extracted (Bobadilla et al., 2013). Depending on the type of content, the approaches for this extraction step vary:

In the case of processing textual information, classic information retrieval techniques can be used to define such attributes automatically (e.g., term frequency, inverse document frequency, and normalization to page length) (Pazzani & Billsus, 2007). As recent advances in natural language processing (NLP) using DNNs have demonstrated the importance of learning good representations for text (Le & Mikolov, 2014), researchers have started to use DNNs in the context of text-based RSs (T. Chen et al., 2017), achieving superior results compared to classical methods.

For visual RSs, early attempts have primarily employed annotated tags (Fan et al., 2008) or low-level visual features (Su et al., 2010), such as color, texture, and shape, to capture the semantics of images. As a result of the success of DNNs in the domain of computer vision (K. He et al., 2016), many recent approaches, like McAuley et al., 2015, R. He and McAuley, 2016, Bracher et al., 2016, Kang et al., 2017, Shankar et al., 2017, and Tuinhof et al., 2018 have now also shifted towards using DNNs for image-based RSs. More specifically, they utilize CNNs as feature extractors for images and integrate these product features into purely content-based or hybrid RSs.

2.7 Adversarial Attacks

Statistical ML and especially, deep learning (DL) have become increasingly popular due to their superior performance in many tasks and are deployed in many real-world applications, ranging from image classification (K. He et al., 2016; Krizhevsky et al., 2012) to speech recognition (G. Hinton, Deng, et al., 2012) and NLP (Devlin et al., 2018). Because of these accomplishments, ML methods have also become an essential tool in many security-sensitive domains, such as spam filtering (Guzella & Caminhas, 2009), malware detection (Sahs & Khan, 2012), and network intrusion detection systems (IDSs) (Yin et al., 2017), where these applications are required to be highly accurate, stable, and reliable, even if underlying data distributions change. These requirements stand in contrast to an attacker's goals, who tries to modify the input data to circumvent these systems. Dalvi et al., 2004 has first demonstrated that machine learning models are often vulnerable to manipulations, causing incorrect classification of adversarial inputs. In particular, neural networks are highly vulnerable to attacks based on small modifications to inputs at prediction time, as first pointed out by Szegedy et al., 2013. More recent research by Grosse et al., 2016 has also proven the existence of adversarial examples for the problem of malware classification. Apart from the security implications, this phenomenon also demonstrates that current models are not learning the underlying concepts in a robust manner (Madry et al., 2017). The following sections will give a definition of adversarial examples, explore different attack methods, discuss the property of transferability and lastly, show what kind of defenses exist.

2.7.1 Adversarial Examples

An adversarial example is a sample of input data that has been modified very slightly in a way that intends to cause a machine learning classifier to misclassify it. In many cases, these modifications can be so subtle that a human observer does not even notice the modification at all, as seen in Figure 2.13, yet the classifier still makes a mistake (Kurakin et al., 2016a).

Currently there is no widely agreed-upon explanation as to why these adversarial examples exist, but rather a variety of explanations were proposed. Some researchers have attributed this phenomenon to the high dimensional nature of the input and parameter space (I. J. Goodfellow et al., 2014), while others have attributed it to the presence of non-robust features in the training data (Ilyas et al., 2019).

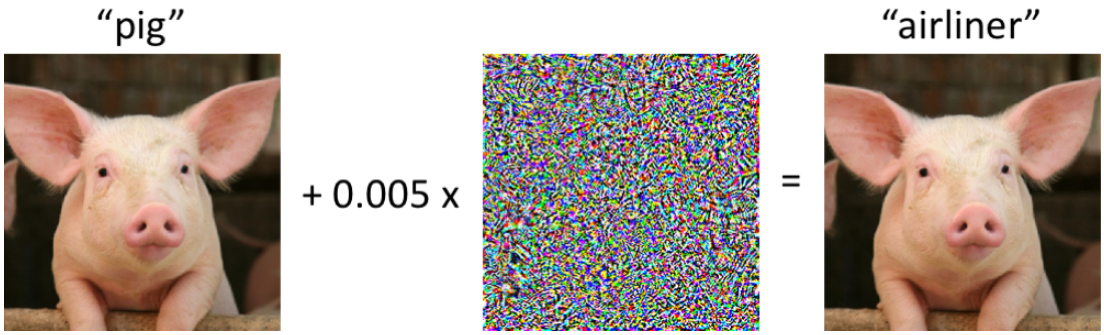


Figure 2.13: On the left, we have an image of a pig that is correctly classified as such by a state-of-the-art CNN. After perturbing the image slightly (every pixel is in the range [0, 1] and changed by at most 0.005), the network now returns class “airliner” with high confidence (Mądry & Schmidt, 2018).

More formally Carlini and Wagner, 2017 defines a targeted adversarial example x' as an example fulfilling the property of $C(x') = t$, where C is a classifier, x an input, y the ground-truth class label and $t \neq y$ the target class, while x' and x are close according to some distance metric.

A less powerful attack also discussed in the literature instead asks for untargeted adversarial examples: instead of classifying x as a given target class, we only search for an input x' so that $C(x') \neq y$ and x, x' are close (Carlini & Wagner, 2017).

There are three widely-used distance metrics in the literature for generating adversarial examples, all of which are l_p norms (Carlini & Wagner, 2017).

1. l_0 distance measures the number of coordinates i such that $x_i \neq x'_i$.
2. l_2 distance measures the standard Euclidean distance between x and x'.
3. l_∞ distance measures the maximum change to any of the coordinates:

$$\|x - x'\|_\infty = \max(|x_1 - x'_1|, ..., |x_n - x'_n|). \tag{2.21}$$

2.7.2 Finding Adversarial Examples

The following sections will describe three well-studied white-box attacks and show how they can be used to find targeted adversarial examples. All approaches share a common theme of using the gradients ∇_x produced by the network with respect to the input for finding adversarial perturbations that optimize for a chosen adversarial goal.

Fast Gradient Sign Method

One of the first and most popular adversarial white-box attacks is the single-step fast gradient sign method (FGSM), initially described by I. J. Goodfellow et al., 2014. Let x be the input to the model, y the label associated with x and $\mathcal{L}(x, y)$ be the loss used

to train the model. We can linearize the loss function around the current value of x, obtaining an ϵ constrained perturbation which can be added to the image to obtain an untargeted adversarial example x' that maximizes $\mathcal{L}$, i.e.

$$x' = x + \epsilon \text{ sign}(\nabla_x \mathcal{L}(x, y)). \tag{2.22}$$

In other words, the attack backpropagates the gradient back to the input data to calculate $\nabla_x \mathcal{L}(x, y)$ and then adjusts the input data by a small step ϵ in the direction that will maximize the loss.

To use FGSM for targeted attacks, trying to maximize the probability for some specific class $t \neq y$, attacking a neural network with CE loss will lead to the following formula for the single-step attack Kurakin et al., 2016b:

$$x' = x - \epsilon \text{ sign}(\nabla_x \mathcal{L}(x, t)). \tag{2.23}$$

Projected Gradient Descent

A rather simple extension of the original FGSM attack is the projected gradient descent (PGD) attack, also known as the basic iterative method (BIM). By iteratively performing the FGSM attack with smaller step sizes α and clipping the resulting perturbations after each iteration, more powerful adversarial examples can be found. More formally a untargeted PGD attack can be written as (Kurakin et al., 2016a):

$$x'_0 = x, \quad x'_{N+1} = Clip_{x,\epsilon}\Big\{x'_N + \alpha \text{ sign}(\nabla_x \mathcal{L}(x'_N, y))\Big\}. \tag{2.24}$$

Similar to FGSM the targeted version looks very similar:

$$x'_0 = x, \quad x'_{N+1} = Clip_{x,\epsilon}\Big\{x'_N - \alpha \text{ sign}(\nabla_x \mathcal{L}(x'_N, t))\Big\}. \tag{2.25}$$

The exact clipping equation, ensuring valid pixel values ($0 \leq x' \leq 1$) and a maximum l_∞-norm perturbation budget ϵ for an adversarial example x', is defined as:

$$Clip_{x,\epsilon}\{x'\} = \min\Big\{1, x + \epsilon, \max\{0, x - \epsilon, x'\}, \Big\} \tag{2.26}$$

where x is the channel values of the original image.

Carlini & Wagner Method

The Carlini & Wagner (CW) attack (Carlini & Wagner, 2017) is one of the most effective white-box attack methods. The core idea of the CW attack is to use an unconstrained optimization formulation and an emperically chosen objective function f. Formally, for

a chosen l_p norm, the problem becomes: given a constant c, and an input image x, find δ that solves

$$\text{minimize} \quad \|\delta\|_p + c \cdot f(x+\delta) \tag{2.27}$$
$$\textit{such that} \quad x + \delta \in [0,1]^n \tag{2.28}$$

To ensure the modification yields a valid image ($0 \leq x_i + \delta_i \leq 1$), known as a box constraint, a change of variables for delta is performed:

$$\delta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i. \tag{2.29}$$

Since $-1 \leq \tanh(w_i) \leq 1$, it follows that $0 \leq x_i + \delta_i \leq 1$, so the solution will automatically be valid. This method enables us to use other optimization algorithms that do not natively support box constraints. Carlini and Wagner, 2017 reported that in their experiments, the Adam (Kingma & Ba, 2014) optimizer was the most effective at quickly finding adversarial examples.

Since the l_∞ distance metric is not fully differentiable, Carlini and Wagner, 2017 found that gradient descent produces very poor results for minimizing the $\|\delta\|_\infty$ term. They instead suggest replacing this term with a penalty for any terms that exceed τ, resulting in the following problem definition for a CW optimizing for $l_\infty \leq \tau$:

$$\underset{\delta}{\text{minimize}} \quad c \cdot f(x+\delta) + \sum_i [(\delta_i - \tau)^+] \tag{2.30}$$

2.7.3 Transferability of Adversarial Examples

An interesting property of adversarial examples discovered by Szegedy et al., 2013 is their *transferability* (also *generalization*). An adversarial example created for one model can sometimes serve as an adversarial example for a different model that has been trained from scratch on the same training set (cross model generalization) or even on a completely disjoint training set (cross training set generalization). This property also enables black-box attacks using surrogate models, which were shown to be effective against public vision application programming intefraces (APIs) (Ilyas et al., 2018).

2.7.4 Defending Against Adversarial Examples

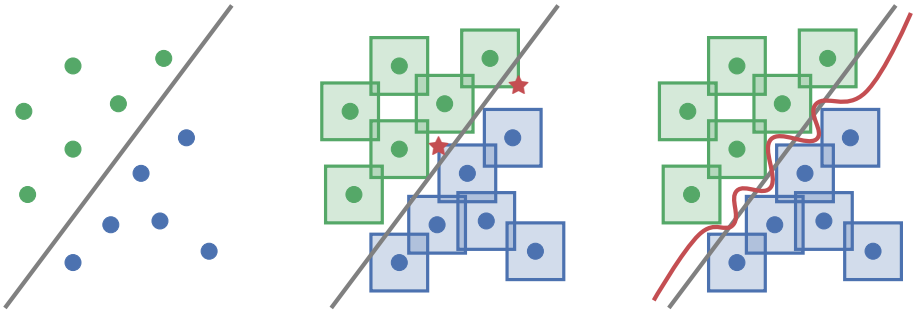
Since the discovery of adversarial examples, numerous studies have been dedicated to finding effective defenses (Lin et al., 2019; Papernot et al., 2016; Tramèr et al., 2017). The class of defenses that have proven to work most reliably (Athalye et al., 2018) is called adversarial training (AT) and is also the class of defenses that we will focus on in this Section. Starting with an explanation of traditional AT we will also describe a suggested improvement called curriculum adversarial training (CAT).

Adversarial Training

The basic idea of AT is to integrate adversarial examples into the optimization procedure of the neural network to train a robust classifier, which can correctly classify adversarial inputs within a defined l_p norm. This changes the problem of training a model to a min-max problem, where our goal is to perform well, no matter what attack an adversary uses. That is, given a dataset $\mathcal{D}$, a parameterized model f_θ and a loss function $\mathcal{L}$, we want to solve the outer minimization problem (Madry et al., 2017):

$$\underset{\theta}{\text{minimize}} \quad \frac{1}{|\mathcal{D}|} \sum_{x,y \in \mathcal{D}} \max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(f_\theta(x+\delta), y). \tag{2.31}$$

Solving the inner optimization can be achieved using any of the previously mentioned attacks but is usually conducted using PGD, due to its favorable trade-off between time-complexity and performance (Madry et al., 2017). A full description of the AT procedure is provided in Algorithm 2. Additionally, Figure 2.14 provides a simplified visualization of the AT problem and shows why classifying examples in a robust way requires a stronger classifier due to a more complicated decision boundary.



(a) A set of points that can be easily separated with a simple decision boundary.

(b) The simple decision boundary does not separate the l_∞-balls (here, squares) around the data points. Hence there are adversarial examples (the red stars) that will be misclassified.

(c) Separating the l_∞-balls requires a significantly more complicated decision boundary. The resulting classifier is robust to adversarial examples with bounded l_∞-norm perturbations.

Figure 2.14: A conceptual illustration of standard vs. adversarial decision boundaries (adapted from Madry et al., 2017)

Algorithm 2 Adversarial Training ($\mathrm{AT}(\mathcal{D}, N, \eta, \mathcal{A})$)

Require: Training data $\mathcal{D}$; Total iterations N; Learning rate η;
Require: An attack $\mathcal{A}$
1: $\theta \leftarrow$ random parameter initialization
2: **for** $i \leftarrow 0$ to N **do**
3: $(x_i, y_i) \leftarrow$ sample batch from training data $\mathcal{D}$
4: $x_i' \leftarrow$ generate adversarial examples using $\mathcal{A}(x_i, y_i)$
5: $\theta \leftarrow$ update parameters using SGD, i.e. $\theta - \eta \sum_i \nabla_\theta \mathcal{L}(f_\theta(x_i'), y_i)$
6: **end for**
7: **return** θ

Curriculum Adversarial Training

A proposed improvement of the traditional AT framework published by Cai et al., 2018 is called curriculum adversarial training. The main idea behind this approach is that generating a curriculum of adversarial examples with gradually increasing attack strengths k, during training can help a model in regards to convergence, leading to higher accuracy results on clean and adversarial inputs. The initial attack strength is set to $k = 0$, causing the normal task to be learned first, before starting with the actual adversarial training. The detailed steps of this method are displayed in Algorithm 3.

Algorithm 3 Curriculum Adversarial Training (Basic)

Require: Training data $\mathcal{D}$; Validation data $\mathcal{V}$; Learning rate η; Epoch iterations n; Maximum attack strength K
Require: A class of attack, denoted as $\mathcal{A}(k)$ whose strength is parameterized by k
1: $\theta \leftarrow$ random parameter initialization
2: **for** $l \leftarrow 0$ to K **do**
3: **repeat**
4: $\theta \leftarrow \mathrm{AT}(\mathcal{D}, n, \eta, \mathcal{A}(l))$
5: **until** $\tilde{l}$-accuracy on $\mathcal{V}$ not increased for 10 epochs
6: **end for**
7: **return** θ

3 Threat Model

To assess the specific vulnerabilities of a content-based filtering (CBF) recommendation system (RS) in the following chapters, we will first perform a systematic analysis of the adversarial goals and capabilities concerning CBF RSs. In this section, we explore the unique threat models for CBF RSs in regards to violations of the CIA triad, depicted in Figure 3.1, a standard model to guide efforts towards securing information systems, declaring confidentiality, integrity, and availability as the primary goals. According to ISO/IEC 27000 (ISO Central Secretary, 2018) these goals are defined as:

- **Confidentiality**: "property that information is not made available or disclosed to unauthorized individuals, entities, or processes".
- **Integrity**: "property of accuracy and completeness".
- **Availability**: "property of being accessible and usable upon demand by an authorized entity".

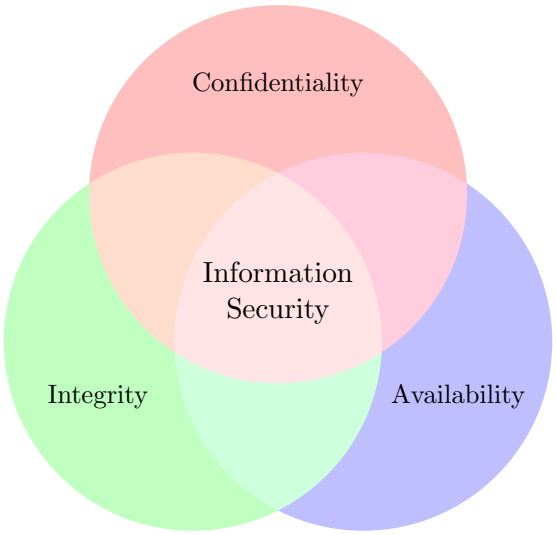


Figure 3.1: The CIA information security triad.

3.1 Content-Based Recommendation System

In this section, we will examine each of the principal goals of the CIA triad in regards to content-based recommendation systems and explore various threats to these principles.

3.1.1 Confidentiality

The confidentiality of the data inside content-based recommendation systems is usually of relatively low significance since it contains no personal data, and all input data for the system, e.g., names, descriptions, categories, and images of items, are usually publicly available through the web service embedding the recommendation system. Therefore we see no unique risks regarding the confidentiality of content-based recommendation systems, and the common risks for web-services apply.

3.1.2 Integrity

Integrity, on the other hand, is essential to the successful deployment of any recommendation system. In the case of content-based recommendation systems, the integrity of predictions is mostly dependent on the quality of the input data, i.e., the item catalog. Figure 3.2 summarizes the potential attacks on the integrity of this data, and the following paragraphs will explain each of the goals and attacks in detail.

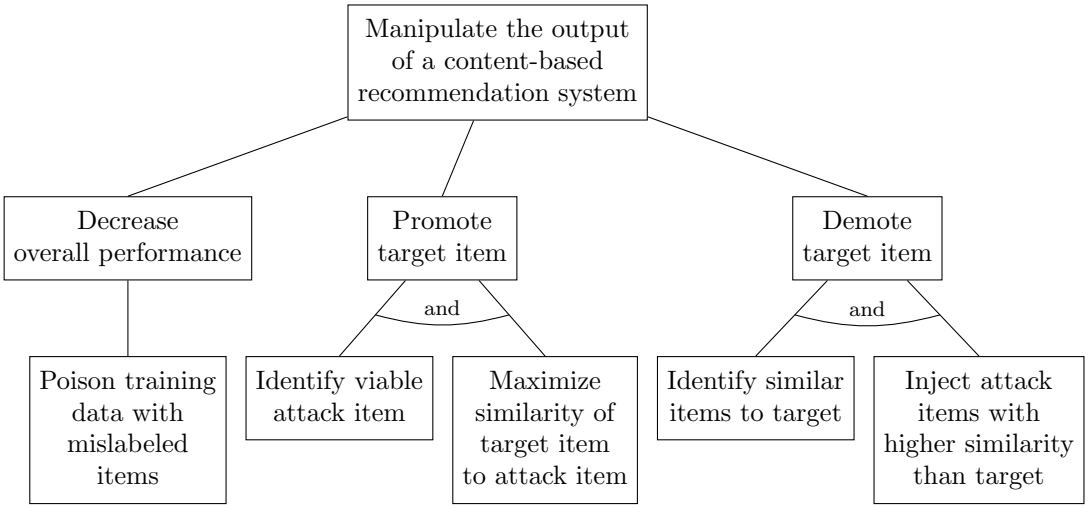


Figure 3.2: Attack tree, summarizing the potential attacks on the integrity of content-based recommendation systems

If the item catalog is accessible and editable by third parties, as in the case of marketplaces or social networks, a defender needs to consider the risk of a malicious data injection. For example, an attacker trying to decrease the overall performance of such a system might try to inject mislabeled or corrupt items into the catalog to confuse the training algorithm and degrade the recommendation system's predictions, as seen in the leftmost leaves of the attack tree in Figure 3.2.

A more targeted attack could attempt to manipulate a target item aiming to maximize its similarity with other popular items to gain more views, as seen in the center of Figure 3.2. To perform such an attack, an adversary needs to know the type of algorithm and also has to have access to the contextual item data used by the algorithm to calculate item similarities. As previously mentioned, this data is usually easily acquirable through the crawling of publicly available web sites. Knowledge about the employed algorithm should also be considered publicly accessible as system security should not depend on the secrecy of the implementation or its components (Scarfone et al., 2008).

Similar to the previous attack goal, an adversary could also attempt to demote a specific target item, as depicted in the right portion of Figure 3.2. In other words, to push it down in the list of recommended articles. We assume that the adversary has no control over the target item in this scenario as it could, for example, be under the control of a competitor. Since the adversary cannot manipulate the target item directly, the only way to demote the target item is to steal its recommendation slots. As the similarity measures used in content-based recommendations are usually symmetrical, the adversary might look at the target's recommendation results to determine its current recommendation slots. After identifying the items on which the target appears as a recommendation, the adversary could inject a set of attack items with a higher similarity than the target item to push it down the list of recommendations and steal its display space.

3.1.3 Availability

As the principal goals of availability and integrity are closely related, similar adversarial goals and risks arise. For example, an adversary trying to increase the overall error rate of the recommendation system will also violate availability at a certain point. Additionally, the universal availability risks for web-services, like a distributed denial-of-service (DDoS), apply as well and should be considered in a complete threat model.

4 Dataset

Throughout this thesis, we work with a public image classification dataset. This dataset is the basis for training the image-based k-NN recommendation system in the following Chapter 5. The following paragraphs will summarize the dataset and explain how we preprocessed the data before training.

4.1 DeepFashion

We use the popular DeepFashion Attribute Prediction [1] dataset published by Liu et al., 2016. This dataset contains 289,222 annotated garment images with their corresponding fashion category and texture type. Refer to Table 4.1 for a full summary of the dataset. Since the type of images is very similar to the ones found in fashion e-commerce sites, this dataset lends itself well to our goal of training a feature extractor for fashion images and is also used by Tuinhof et al., 2018 in the reference implementation of the model, which we are trying to reproduce, attack and defend in the following chapters.

	Classification		Samples			
Dataset	Type	No.	Total	Train	Val	Test
DeepFashion Category	Multinomial	46	289,222	209,222	40,000	40,000
DeepFashion Texture	Multinomial	156	111,405	80,522	15,538	15,345

Table 4.1: Summary of the original DeepFashion Attribute Prediction dataset.

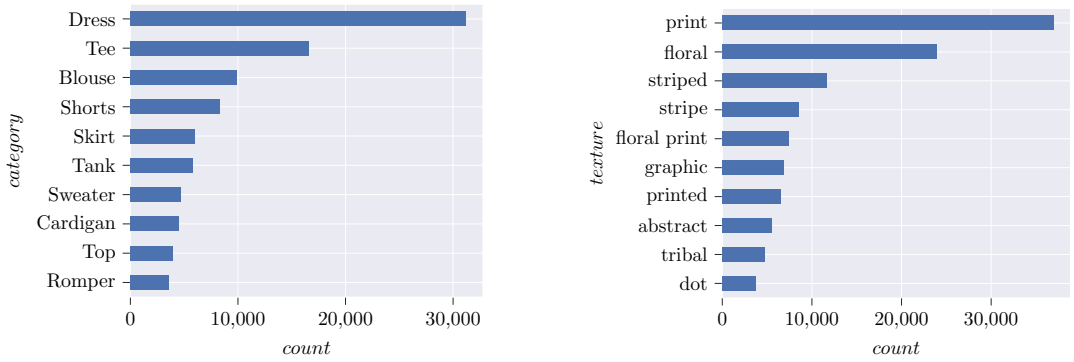
The original DeepFashion dataset contains quite a few duplicate entries, which we filtered out by deduplicating examples using perceptual hashing (Jain et al., 2019). The resulting dataset is summarized in Table 4.2. In order to stabilize the training procedure, we also removed all examples without texture labels arriving at a total number of 106,649 labeled example images.

	Classification		Samples			
Dataset	Type	No.	Total	Train	Val	Test
DeepFashion Category	Multinomial	46	279,057	201,908	38,620	38,529
DeepFashion Texture	Multinomial	156	106,649	77,059	14,835	14,755

Table 4.2: Summary of the preprocessed DeepFashion Attribute Prediction dataset.

[1] http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion/AttributePrediction.html

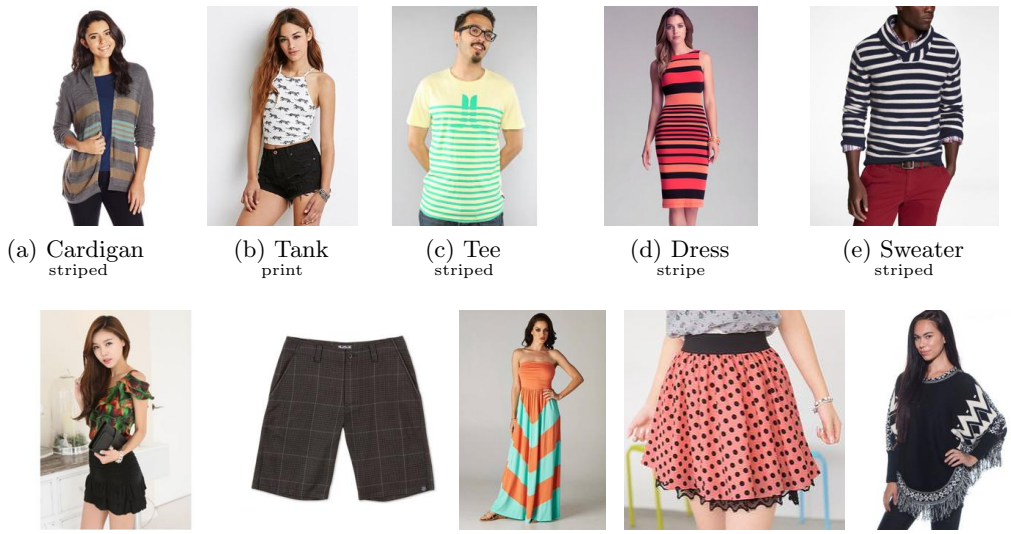
In Figure 4.1, we provide an overview of the top ten classes for both classification tasks. Additionally, we display ten randomly selected sample images from the DeepFashion dataset with their corresponding labels in Figure 4.2.



(a) Example counts of the Top-10 category labels

(b) Example counts of the Top-10 texture labels

Figure 4.1: Example counts for the most common labels of the DeepFashion dataset.



(a) Cardigan striped

(b) Tank print

(c) Tee striped

(d) Dress stripe

(e) Sweater striped

(f) Blouse floral

(g) Shorts houndstooth

(h) Dress chevron

(i) Skirt dotted

(j) Poncho tribal

Figure 4.2: Randomly sampled images from the DeepFashion dataset.

5 Model

In this chapter, we will define the type of image-based recommendation system that we used as the foundation for our subsequent robustness studies. More specifically, we will provide a detailed description and evaluation of our adopted two-stage model, based on the work of Tuinhof et al., 2018.

5.1 Image-Based Recommendation System

We chose to reproduce the model, published by Tuinhof et al., 2018, for our following experiments, as it relies purely on content-based data and utilizes the publicly available DeepFashion dataset. The proposed approach consists of two stages: a convolutional neural network (CNN) fashion classifier, predicting category and texture of garments, and a similarity search using a k-nearest neighbors (k-NN) algorithm, based on the computed latent space feature embeddings from the classifier. This rather simple technique proves successful at finding similar items based on the texture and category of a supplied input image.

Due to a non-disclosure agreement, signed by the authors of the original paper, it was unfortunately not possible to retrieve their source code or model weights. Therefore, we had to reproduce the results from scratch. In the following subsections, we will document our reproduced results, and describe some minor improvements.

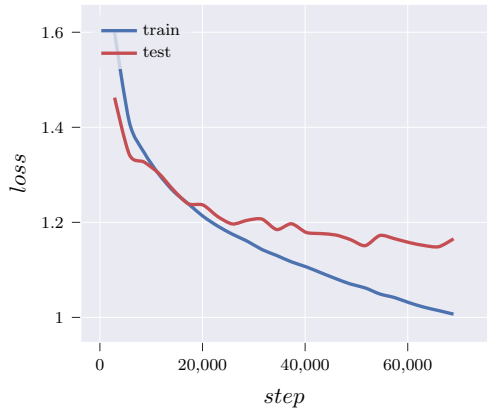
5.1.1 Convolutional Deep Fashion Classifier

In the first step, we train a CNN to predict the category and texture type of fashion garments simultaneously. The authors of the original paper trained two individual networks for each task and concatenated both networks' embeddings for the second-stage k-NN search. Since studies have shown that multi-task learning can lead to significant performance improvements (Zhang & Yang, 2017), we decided to evaluate training a network on both tasks at once, using two independent fully connected softmax layers, instead of one, as a prediction head. Formally, our multi-task model can be written as

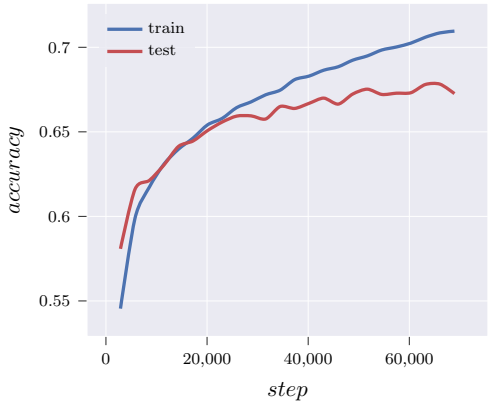
$$\mathcal{N}(\mathbb{W}, \boldsymbol{X}) = [\mathcal{S}(\mathbb{C}, \mathcal{F}(\mathbb{I}, \boldsymbol{X})), \mathcal{S}(\mathbb{T}, \mathcal{F}(\mathbb{I}, \boldsymbol{X}))], \tag{5.1}$$

where $\mathbb{W} = (\mathbb{I}, \mathbb{C}, \mathbb{T})$ are weight matrices, $\mathcal{S}(\mathbb{C}, \boldsymbol{X})$ and $\mathcal{S}(\mathbb{T}, \boldsymbol{X})$ are fully connected softmax output layers that perform the classifications for category and texture, and $\mathcal{F}(\mathbb{I}, \boldsymbol{X})$ is the shared CNN backbone, which is used as a feature extractor.

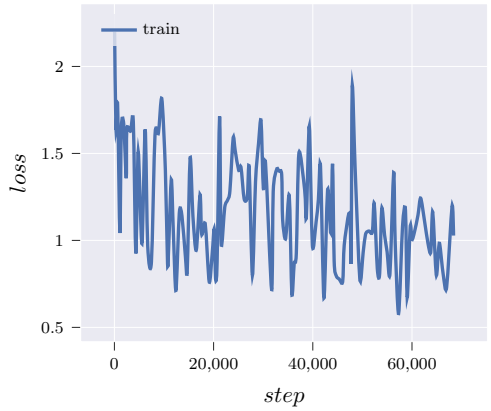
The original paper evaluated two state-of-the-art CNN architectures at the time of publication– Inception with batch norm and AlexNet. Since then, many more efficient architectures, like ShuffleNet (Ma et al., 2018) and MobileNetV2 (Sandler et al., 2018), have been proposed. Due to its superior performance, we chose to use MobileNetV2 as our architecture for the CNN backbone. As an initialization for the weights, we used the weights of a pre-trained network on ImageNet [1]. This way of initializing networks is called transfer learning and has proven to be beneficial for convergence (Shin et al., 2016). For training the MobileNetV2, we use the Adam (Kingma & Ba, 2014) optimizer with a batch size of 32, and a learning rate of 0.001. Our training history can be seen in Figure 5.1 and our final evaluation results on the test set in Table 5.1



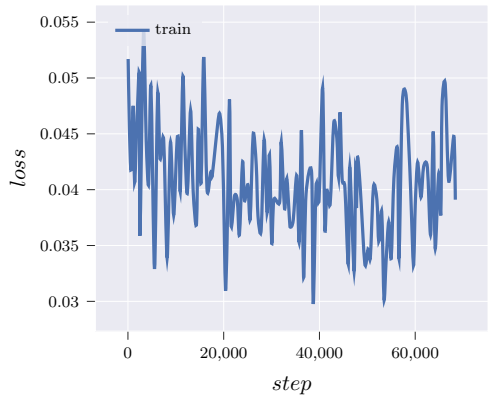
(a) Plot of combined model loss on training and test dataset



(b) Plot of category accuracy on training and test dataset



(c) Plot of category classification loss on training dataset



(d) Plot of texture classification loss on training dataset

Figure 5.1: Training history of our fashion classifier trained for 24 epochs

[1] Pre-Trained MobileNetV2 https://pytorch.org/hub/pytorch_vision_mobilenet_v2/

Category	Ours	Tuinhof et al., 2018
Accuracy	68.25	63.00
Top-5 Accuracy	93.14	84.00
CE-Loss	1.09	1.27

(a) Our category classifier results in comparison to the results reported in the original paper by Tuinhof et al., 2018.

Texture	Ours
Top-1 Precision	43.68
BCE-Loss	0.03

(b) Our texture classifier results.

Table 5.1: Evaluation results on test set for our multi-task *DeepFashion* classifier.

As seen in Table 5.1a, we slightly outperform the original accuracy metric on the category classification task by about 5%. The authors of the DeepFashion dataset formulated the second task of texture classification as a multi-label problem. Tuinhof et al., 2018, on the other hand, chose to reformulate it as a single label task, by randomly sampling one label from the set of correct labels. We chose to treat the texture label as a multi-label task and make use of the entire dataset. Therefore, our results for the texture classifier are not directly comparable to Tuinhof et al., 2018, as the metrics and losses used in single and multi-label settings differ. As a loss function for the texture classifier, we used the binary cross-entropy (BCE) function, which performs elementwise sigmoid activations instead of a softmax activation, as in the case of categorical cross-entropy (CE). Our results in Table 5.1b show that our network has been reasonably successful at learning the task of texture classification, achieving almost 44% precision on the top predicted texture from 156 possible classes. We also applied several standard image augmentation techniques to increase the dataset size effectively and prevent overfitting. These include random rotations with a maximum rotation angle of $\pm 15°$, random rescaling, and cropping within a range of [75%, 125%] and random horizontal flips.

5.1.2 Ranking in Feature Space using k-NN Search

The CNN backbone is used as a feature extractor, and returns feature vectors $\mathcal{F}(\boldsymbol{X})$ of size $d = 1280$ for any input image. This feature extractor is applied to all $n = 106,649$ images from the dataset, as described in Chapter 4. The resulting vectors are stacked to obtain a $n \times d$ article embedding matrix, which we use for a k-NN ranking algorithm.

$$sim(A, B) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} \tag{5.2}$$

$$dist(A, B) = 1 - sim(A, B) \tag{5.3}$$

As a similarity measure, we use the cosine distance, defined in Equation 5.3, which is commonly used in recommendation systems (Desrosiers & Karypis, 2011). To improve the performance of the k-NN search, we additionally use an approximate nearest neighbor index[2], that uses a highly optimized data structure, called hierarchical navigable small world (HNSW) graphs, which was originally published by Malkov and Yashunin, 2018.

[2]NMSLIB https://github.com/nmslib/nmslib

The resulting embedding space of articles is visualized in Figure 5.2. Similar categories and textures are grouped due to their proximity in feature space, which indicates the usefulness of our generated embeddings for finding similar articles. The ranked nearest neighbors for two randomly chosen articles can be seen in Figure 5.3. Subjectively, the recommendations indeed look quite similar to the query images.

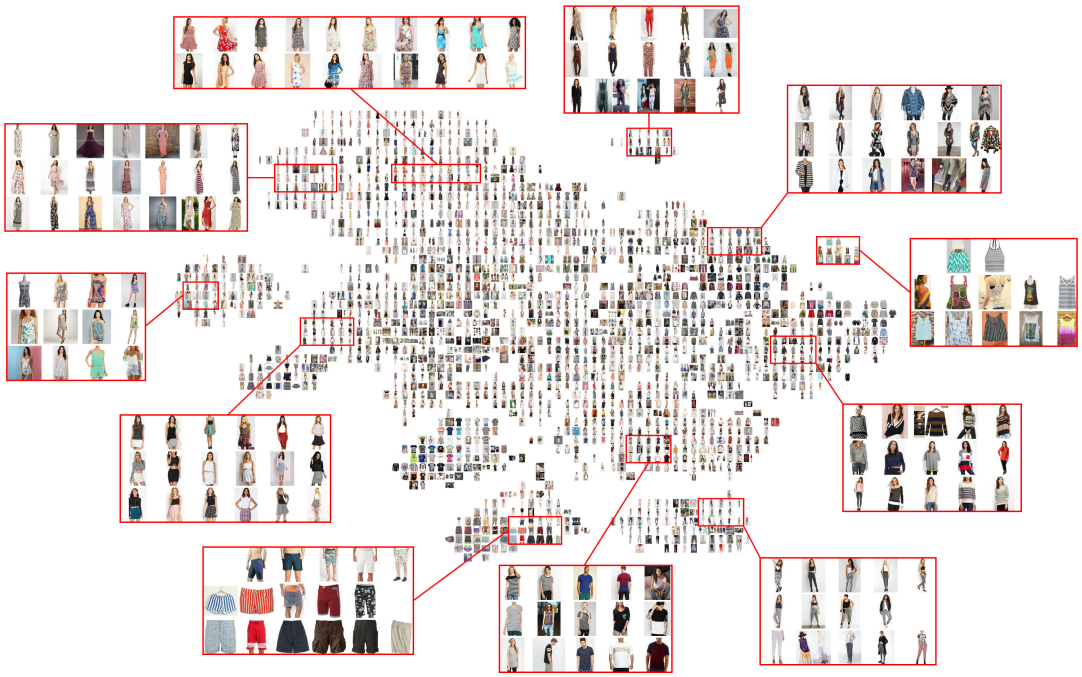


Figure 5.2: t-SNE visualization of articles from the DeepFashion dataset, using their feature vectors from the penultimate layer of the classifier trained in Section 5.1.1. For clarity, the space is discretized into a grid and for each grid cell one image is randomly selected among overlapping instances.

Figure 5.3: Ranked k-NN results for two randomly selected items

6 Attacks

After giving an overview of the dataset and model, which we use as a basis for our following experiments, we will now define our proposed targeted item-to-item attacks and discuss our experimental results against an undefended visual recommendation system, as described in Chapter 5.

6.1 Adversary Model

Before diving into the details of our attack experiments, we outline our adversary threat model based on the guidelines proposed by Carlini and Wagner, 2017. Our adversary's assumptions are:

- **adversary goal:** The adversary is interested in minimizing the cosine distance, as defined in Equation 5.3, between the latent-space embeddings of an attack article image to a pre-existing target article image. By minimizing this distance, the chosen attack article decreases its rank in the list of nearest neighbors of the target article, thereby promoting the attack article.
- **adversary knowledge:** We assume a white-box knowledge setting, in which the adversary holds full knowledge of the feature extraction model parameters used to estimate the targeted perturbation.
- **adversary capability:** We restrict the adversary capability to make l_∞-norm constrained perturbations to the attack image.

6.2 Setup

A visualization of our proposed item-to-item attack setup is depicted in Figure 6.1. We first perform a forward pass of the convolutional neural network feature extractor $\mathcal{F}$ for both the original attack image A and our target image T. Using the resulting feature embeddings, we can then calculate the latent space cosine distance from Equation 5.3, between our unmodified attack and target images. We choose this distance-metric as our adversary objective function which we want to minimize while restricting the l_∞-norm of our additive perturbations δ, i.e.

$$\underset{\|\delta\|_\infty \leq \epsilon}{\text{minimize}} \quad dist(\mathcal{F}(A+\delta),\, \mathcal{F}(T)). \tag{6.1}$$

Using the three evaluated gradient-based attack methods, namely fast gradient sign method (FGSM), projected gradient descent (PGD), and Carlini & Wagner (CW), we

minimize this objective function by performing a backward pass through the network to the attack image and adjusting the targeted perturbations in the opposite direction of the calculated gradients. Depending on the attack method, we repeat this optimization step iteratively to approximate the loss surface as accurately as possible.

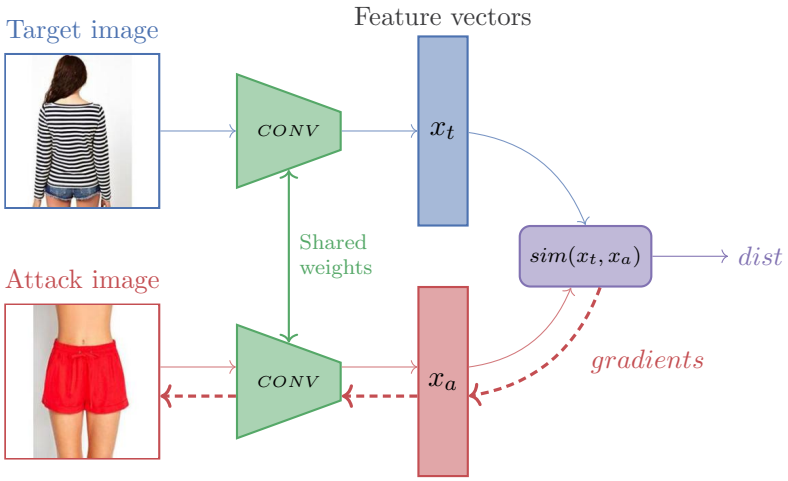


Figure 6.1: Graph visualization of our white-box attack setup, used for exploiting the content-based recommendation system, described in Chapter 5. The goal of the attack is to perturb the image pixels of a given attack article, in order to minimize its feature space cosine distance to a chosen target article, while keeping the applied perturbation within a defined l_∞ budget.

6.3 Evaluation Metric

To compare the effectiveness of the different attack methods, we define a quantitative evaluation metric, measuring the success of an adversarial example in achieving the adversary goal, defined in Section 6.1. We evaluate these success metrics on n random attack tuples sampled from the test set of article images for each attack. To increase the difficulty of attacks and prevent false positives, we ensure that items of each attack tuple belong to different garment categories. We consider an attack successful if the adversarial article can decrease its recommendation rank $rank(\mathcal{F}(A+\delta), \mathcal{F}(T)$ among the k-nearest neighbors (k-NN) for the target under a defined minimum threshold $rank_{min}$. Therefore the success rate calculated over n attack tuples is defined as

$$success\,rate = \frac{1}{n}\sum_{i=0}^{n} \mathbb{1}\big\{rank(\mathcal{F}(A_i+\delta_i),\, \mathcal{F}(T_i)) \leq rank_{min}\big\}. \tag{6.2}$$

Depending on the environment and deployment of the recommendation algorithm, the minimum rank required to gain display space might vary. Therefore we compare the success rates for multiple rank thresholds, calculating the number of successful attacks that achieve a target rank equal to or lower than $rank_{min} = (1, 3, 10, 100)$.

6.4 Evaluation of Attack Methods

In this section, we will discuss our experimental results for performing our proposed targeted item-to-item attacks against our recommendation system using three well-known white-box attack methods, namely FGSM, PGD, and CW.

6.4.1 Fast Gradient Sign Method

Starting with the single-step fast gradient sign method, we assess its effectiveness in minimizing the cosine distances between image embedding produced by our convolutional neural network (CNN) feature extractor. An example attack tuple calculated using FGSM can be seen in Figure 6.2.

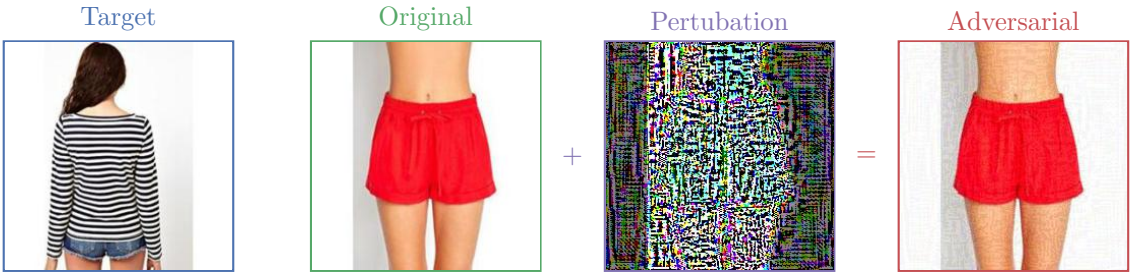


Figure 6.2: Adversarial example, created using the FGSM with $\epsilon = 0.03$. The perturbation is normalized for visualization purposes.

The cosine distance of feature vectors before and after the attack for this example are:

$$dist(\mathcal{F}(A), \mathcal{F}(T)) = 0.6247 \tag{6.3}$$

$$dist(\mathcal{F}(A + \delta), \mathcal{F}(T)) = 0.5267 \tag{6.4}$$

Already from just looking at this one example, we can see that the achieved reduction in the cosine distance after the attack is quite limited and is not enough to achieve a significant rank within the nearest neighbors of the target article. As seen in Table 6.1, this first impression is confirmed when we evaluate the success rates achieved by FGSM on a broader set of attack tuples for ϵ values ranging from 0.01 to 0.05.

	Maximal Perturbation				
$rank_{min}$	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$
1	0.12	0.07	0.06	0.02	0.01
3	0.27	0.16	0.14	0.09	0.07
10	0.64	0.44	0.32	0.18	0.13
100	2.87	2.45	1.83	1.36	0.99

Table 6.1: Success rates (%) calculated over 10,000 random attack tuples using FGSM.

The maximum success rate achieved using FGSM for $rank_{min} = 3$ in our experiments is 0.27%, indicating that the one-step optimization step is not sufficient for successfully finding worst-case perturbations that fulfill our adversary objective. The fact that the performance decreases with larger step sizes (higher ϵ values) reinforces this hypothesis.

To further illustrate the effect that our evaluated attacks have on the cosine distances of attack tuples, we introduce a scatter plot with a fitted quantile regression of the cosine distances before and after each attack. This plot can help us better understand the strengths and weaknesses of our attacks and defenses. For a perfectly robust model, all points should be located on this plot's identity (green dashed line). On the other hand, a perfect attack should be able to push all points toward the x-axis of this plot, minimizing the distances of all attack tuples to values very close to 0.0.

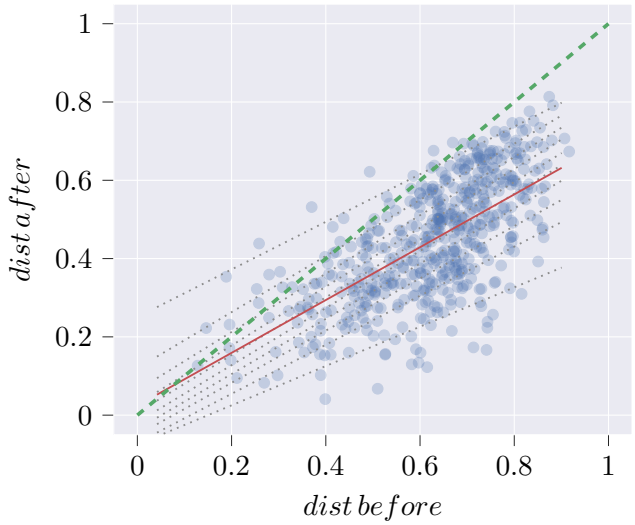


Figure 6.3: Quantile regression plot of 512 sampled cosine distances between target and attack article, before and after performing FGSM attacks, using $\epsilon = 0.05$

When attacking our undefended model using the FGSM attack with $\epsilon = 0.05$, we see a pretty mixed result in Figure 6.3. While the attack reduced some distances, many other distances increased after the attack, indicating failed approximations of the actual loss surface. This plot confirms our previous findings, indicating that the FGSM attack is unsuitable for our targeted adversary objective.

6.4.2 Projected Gradient Descent

For projected gradient descent we use a step size of $\alpha = \frac{\epsilon}{k}$ where k is the number of total iterations. In the following section, we assess the effectiveness of the iterative projected gradient descent method using $k = (8, 16, 32, 64, 128)$ for optimizing our adversarial goal. An example attack tuple calculated using PGD can be seen in Figure 6.4.

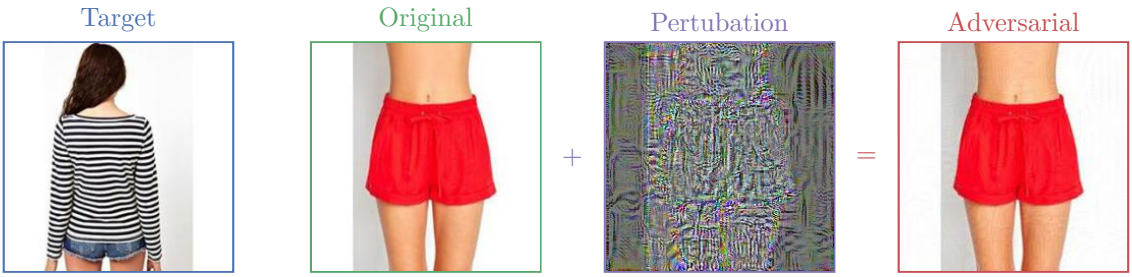


Figure 6.4: Adversarial example, created using PGD with $\epsilon = 0.03$ and 32 iterations. The perturbation is normalized for visualization purposes.

The cosine distance of feature vectors before and after the attack for this example are:

$$dist(\mathcal{F}(A), \mathcal{F}(T)) = 0.6247 \tag{6.5}$$

$$dist(\mathcal{F}(A + \delta), \mathcal{F}(T))) = 0.0500 \tag{6.6}$$

Judging by this one example, we can already see that the achieved reduction in the cosine distance after the attack is significantly higher than in the case of FGSM and is high enough to successfully rank on the first place among the nearest neighbors of the target article. The effect that a successful injection of this adversarial example into the product catalog would have on the resulting recommendations for the target article is shown in Figure 6.5. As seen in Table 6.2, this first impression is confirmed once again when we evaluate the success rates achieved by PGD with 64 iterations on a broader set of attack tuples for ϵ values ranging from 0.01 to 0.05.

Figure 6.5: Ranked recommendation results for original k-NN index (top) and manipulated index with injected PGD adversarial example (bottom)

	Maximal Perturbation				
$rank_{min}$	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$
1	36.44	77.81	86.81	89.65	91.02
3	44.33	86.40	94.06	96.13	97.09
10	50.21	89.61	95.90	97.56	98.22
100	62.55	94.13	97.95	98.74	99.13

Table 6.2: Success rates (%) calculated over 10,000 random attack tuples using PGD-64.

The maximum success rate for $rank_{min} = 3$ achieved using PGD with 64 iterations in our experiments is 97.09%, indicating that an iterative optimization approach, like PGD, is significantly more effective at successfully finding worst-case perturbations that fulfill our adversary objective. These results are consistent with the scientific consensus that iterative optimization-based attacks are strictly stronger than single-step attacks and should achieve strictly superior performance in a white-box setting (Athalye et al., 2018). When we look at the impact on the cosine distances in Figure 6.6, we see a drastic improvement compared to FGSM. Almost all points were successfully pushed onto the x-axis. Nonetheless, there is still room for improvement, as the spread of points with higher initial distances increases significantly.

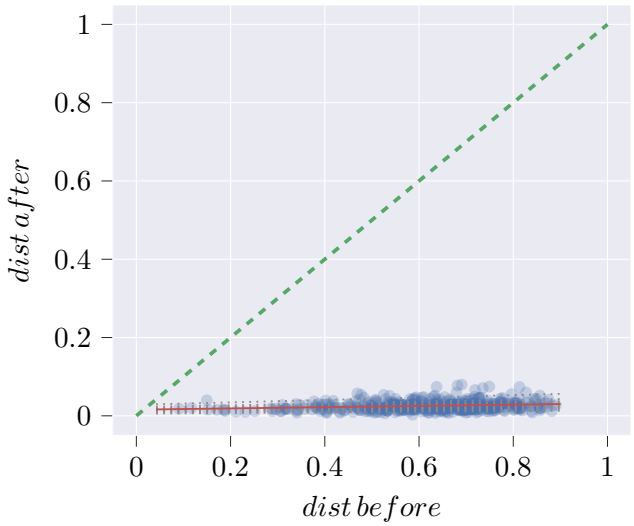


Figure 6.6: Quantile regression plot of 512 sampled cosine distances between target and attack article, before and after performing PGD-32 attacks, using $\epsilon = 0.05$

6.4.3 Carlini & Wagner Method

Finishing off with the strongest and most costly of our evaluated attacks, we implemented and tested the state-of-the-art CW attack method for our adversary objective. Instead of maximizing for a misclassification, we replace the corresponding loss term with our distance metric defined in 6.1. The original implementation uses an iterative search to

find the smallest possible perturbation, that fulfills the adversary objective. However, since we do not care about finding the smallest perturbation and only care about staying within the defined l_∞-norm constraints, we used a fixed value for $\tau = \epsilon$. Using the Adam optimizer with a learning rate of 0.005, we perform 1,000 optimization steps. An example attack tuple calculated using CW can be seen in Figure 6.7.

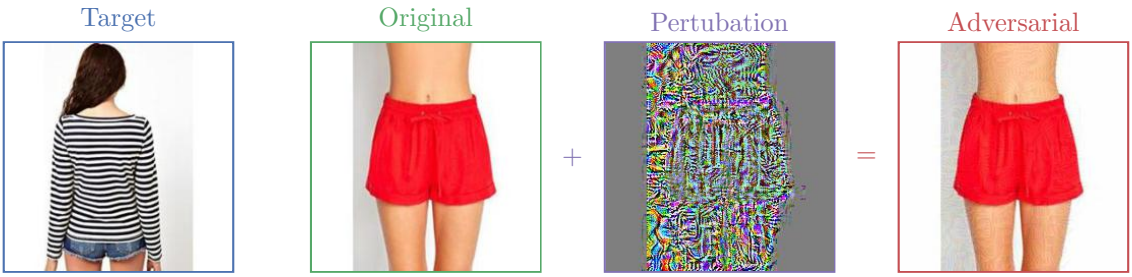


Figure 6.7: Adversarial example, created using the CW method with $\epsilon = 0.03$ and 1,000 iterations. The perturbation is normalized for visualization purposes.

Cosine distance of feature vectors before and after attack:

$$dist(\mathcal{F}(A), \mathcal{F}(T)) = 0.6247 \tag{6.7}$$

$$dist(\mathcal{F}(A + \delta), \mathcal{F}(T)) = 0.0049 \tag{6.8}$$

Examining this one example, we can see that the achieved reduction in the cosine distance after the attack is even higher than in the case of PGD and is therefore also high enough to successfully rank on the first place among the nearest neighbors of the target article, as seen in Figure 6.8. As seen in Table 6.3, this first impression is confirmed once again when we evaluate the success rates achieved by CW with 1,000 iterations on a broader set of attack tuples for ϵ values ranging from 0.01 to 0.05.

Figure 6.8: Ranked recommendation results for original k-NN index (top) and manipulated index with injected CW adversarial example (bottom)

	Maximal Perturbation				
$rank_{min}$	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$
1	74.60	94.10	96.40	97.60	97.80
3	83.10	98.10	99.40	99.70	99.70
10	86.60	98.40	99.50	99.90	99.90
100	91.30	99.40	99.90	100.00	100.00

Table 6.3: Success rates (%) calculated over 1,000 random attack tuples using CW-1000.

The maximum success rate for $rank_{min} = 3$ achieved using CW with 1,000 iterations in our experiments is 99.70%, indicating that the more sophisticated approach of CW, is indeed most effective at successfully finding worst-case perturbations that fulfill our adversary objective. When we look at the impact on the cosine distances in Figure 6.9, we see a nearly perfect result. The vast majority of points were successfully pushed onto the x-axis. Additionally, the spread of points along the y-axis is significantly narrower than in the case of PGD, indicating a stronger and more reliable attack.

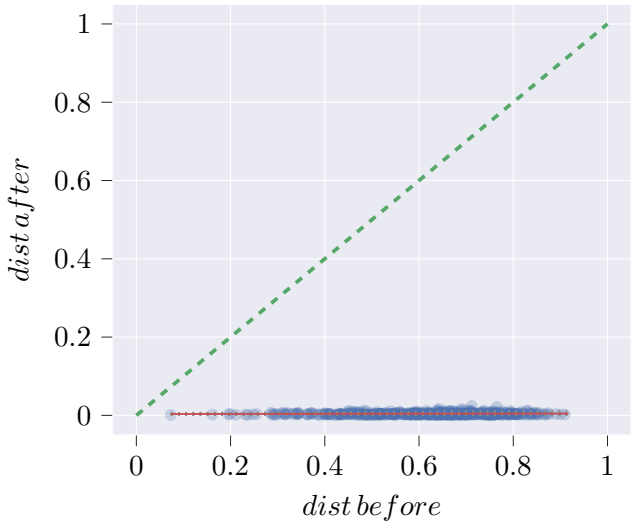


Figure 6.9: Quantile regression plot of 512 sampled cosine distances between target and attack article, before and after performing CW-1000 attacks, using $\epsilon = 0.05$

6.4.4 Comparison

Finally, we want to compare the achieved results of all evaluated attack methods. An overview of the different attack success rates for a minimum target rank of 3 or lower is shown in Table 6.4 and Figure 6.10. Looking at the results, PGD and CW achieve remarkably high success rates going up to 99.70%. FGSM, on the other hand, achieved insufficient results and was not able to achieve our adversary objective. When attacking an undefended model, the performance increase of CW-1000 compared to PGD-128 is relatively small and arguably not worth the extra computational cost (approx. $\times 7.5$),

which the significantly more complex attack method requires. Therefore we consider PGD-128 as the best trade-off between attack strength and computational cost.

	Maximal Perturbation				
Attack	$\epsilon = 0.01$	$\epsilon = 0.02$	$\epsilon = 0.03$	$\epsilon = 0.04$	$\epsilon = 0.05$
FGSM	0.27	0.16	0.14	0.09	0.07
PGD-8	22.19	36.40	35.60	32.41	27.95
PGD-16	33.36	64.28	72.95	74.06	72.69
PGD-32	40.93	78.95	88.85	91.80	92.78
PGD-64	44.33	86.40	94.06	96.13	97.09
PGD-128	45.92	89.79	96.33	97.69	98.32
CW-1000	83.10	98.10	99.40	99.70	99.70

Table 6.4: Success rates (%) for $rank_{min} = 3$, calculated over 10,000 random article tuples (1,000 in the case of CW) for all evaluated attacks and ϵ values.

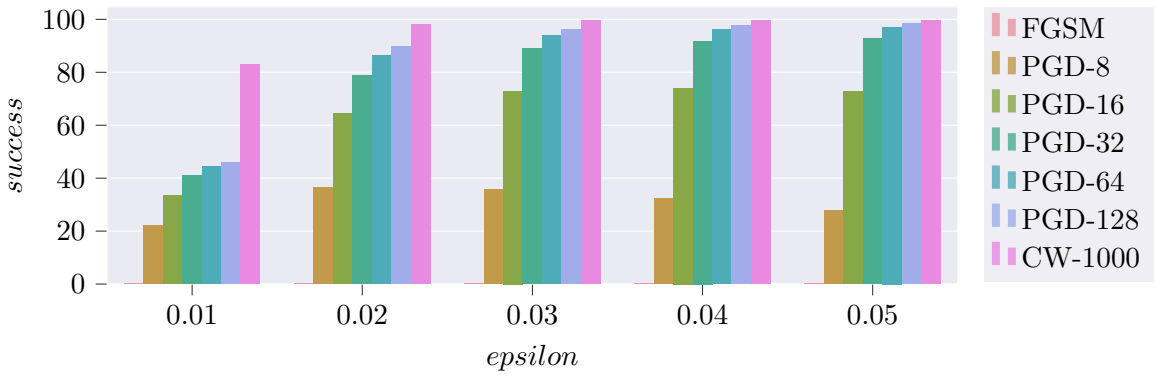


Figure 6.10: Success rates (%) for $rank_{min} = 3$, calculated over 10,000 random article tuples (1,000 in the case of CW) for all evaluated attacks and ϵ values.

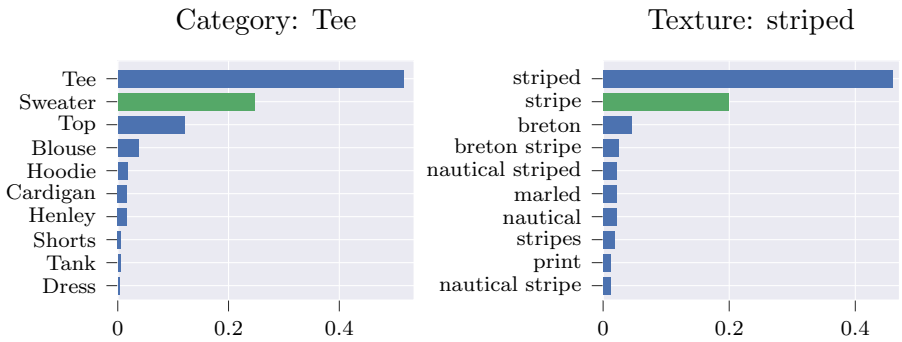
6.4.5 Classifier Impact

An interesting observation we made during our experiments, is that the performed attacks on the cosine distance of image embeddings also indirectly attack the classifier, which was initially trained in Chapter 5. By minimizing the cosine distance between two articles, the predictions of the classifier for these articles also become more similar, as seen in Figure 6.11.

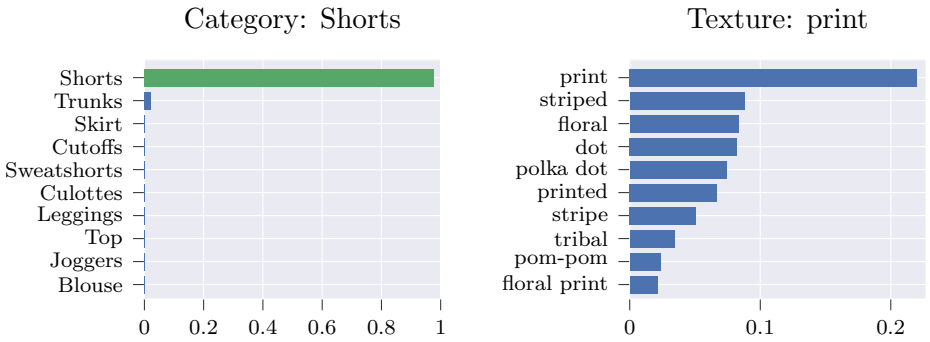
(a) Target image with ground-truth labels: *category* = *Sweater*, *texture* = *stripe*.

(b) Original image with ground-truth labels: *category* = *Shorts*, *texture* = *linen*.

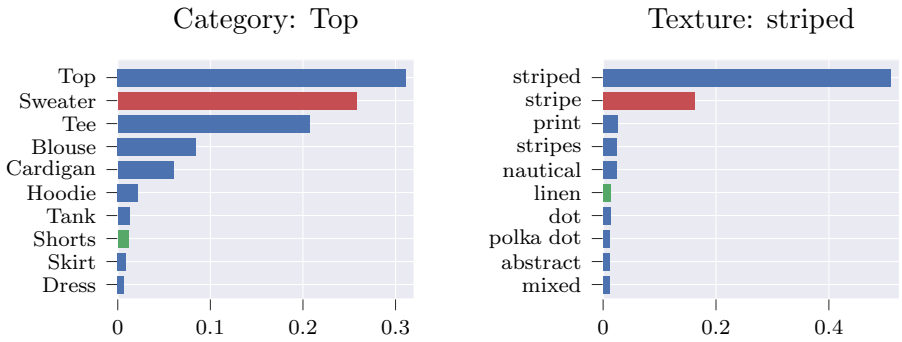
(c) Adversarial image, created using PGD with 32 iterations and $\epsilon = 0.03$.



(d) Classifier predictions for category and texture of target image.



(e) Classifier predictions for category and texture of original image.



(f) Classifier predictions for category and texture of adversarial image.

Figure 6.11: k-NN attacks indirectly attack classifier.

7 Defenses

In the previous chapter, we successfully performed several targeted item-to-item attacks that allow an attacker to promote their items by adding almost imperceptible perturbations to product images. These findings raise a fundamental question:

How can we defend our recommendation system against adversarial inputs?

There is now a sizable body of work proposing various defense mechanisms against adversarial examples. Even though researchers have published many different defense techniques, the research area is still relatively young (Szegedy et al., 2013), and new studies frequently circumvent a lot of published defenses. A prominent example of such a study was published by Athalye et al., 2018. They evaluated nine non-certified white-box-secure defenses published at the ICLR 2018 conference and identified obfuscated gradients, a phenomenon that leads to a false sense of security in defenses against adversarial examples. Out of the nine evaluated defenses, they circumvented six completely, and one partially. Two defenses were considered useful, and are both based on the idea of adversarial training (AT). Hence we also based our experiments on defenses belonging to this class. We evaluate using these defense techniques to train our convolutional neural network (CNN) feature extractor in the following sections and show the impact these defenses have on our proposed attacks against our k-nearest neighbors (k-NN) recommendation system.

7.1 Adversarial Training

Starting with vanilla adversarial training, we study the approach of Madry et al., 2017. During the adversarial training procedure, we instantiate the attack used of our CNN feature extractor with projected gradient descent (PGD) using eight iterations and restricting l_∞ perturbations to $\epsilon = 0.03$. In contrast to our item-to-item attacks performed in Chapter 6, our adversary objective during adversarial training, is to increase the likelihood of misclassification for the category, and texture attributes. We perform the adversarial training for 24 epochs, as we did for our regular classifier.

Looking at the evaluation metrics for our classifier in Table 7.1, we can see that, as expected, our performed attacks reduced our regular classifier performance drastically, while our adversarially trained classifier was able to learn a more robust decision boundary. This increase in robustness is traded in for a decrease in accuracy on clean images. In our case, the accuracy for the adversarially trained model in predicting the correct garment category dropped by 12.19% on clean images, while we gained 48.69% for the same task on adversarial images. Similar results can be seen for the multi-class

task of texture classification. This trade-off is well-known and has been attributed to the fundamental conflict between the goal of adversarial robustness and that of standard generalization (Tsipras et al., 2018).

Category	Adversarial	Regular	Δ
Clean Accuracy	56.06	68.25	− 12.19
Adversarial Accuracy	48.71	0.02	+ 48.69

(a) Category classification results.

Texture	Adversarial	Regular	Δ
Clean Top-1 Precision	39.58	43.68	− 4.10
Adversarial Top-1 Precision	38.60	19.51	+ 19.09

(b) Texture classification results.

Table 7.1: Comparison of our evaluation results on a clean and adversarial test set for a adversarially trained and regular classifier. The adversarial test set was generated using the PGD attack method with eight iterations and $\epsilon = 0.03$.

Since we are interested in evaluating if this kind of training procedure can help us mitigate the attacks on our k-NN recommendation system, we evaluate the success rates of our previously proposed attacks against our adversarially trained feature extractor. To get a first impression, we try to reproduce our targeted attacks on the same attack tuple that we used to demonstrate our attacks in the previous chapter. With relatively low epsilon values up to 0.05, which were enough to reach 99.70% attack success rate for our undefended model, as seen in Table 6.4, we fail to achieve a significant rank with our adversarial examples. Even with unrealistically high epsilon values like 0.3, we still fail to rank under the ten nearest neighbors, as seen in Figure 7.1. Interestingly, for the adversarially trained model, the adversarial images with high epsilon values start to show relevant features of the target image. For example, in Figure 7.2, the stripes of the striped sweater start to appear in the adversarial image. This phenomenon does not occur for our regular model and might indicate that our adversarially trained model learned more relevant features, which help it defend against unrealistic manipulations of the input.

Figure 7.1: A recommendation result of our adversarially trained model after a targeted attack. The adversarial example generated using the Carlini & Wagner (CW) method for $\epsilon = 0.3$, which we injected into the product catalog ranks on place 39 and is therefore not visible in the nearest neighbors displayed above.

(a) $\epsilon = 0.01$ (b) $\epsilon = 0.05$ (c) $\epsilon = 0.1$ (d) $\epsilon = 0.2$ (e) $\epsilon = 0.3$

Figure 7.2: Adversarial examples generated using CW-1000 for our adversarially trained recommendation system with increasing ϵ values ranging from 0.01 to 0.3. The target item for the attack is the same striped sweater as in Chapter 6.

The first impression of the robustness characteristics regarding our proposed attacks is confirmed, when we look at the empirical evaluation results over a broader set of article tuples in Table 7.2 and Figure 7.3. Using adversarial training, we were able to reduce the attack success rate for a perturbation budget of $\epsilon = 0.05$ from 99.70% to a mere 0.30%. When we look at higher and more noticeable perturbation budgets $\epsilon \geq 0.1$, the success rates start to increase slowly. The CW method once again strongly outperforms all other tested methods, demonstrating its superior effectiveness at navigating the loss surface of adversarial examples. We conclude that the achieved reduction in success rates for realistic ϵ values ≤ 0.05, makes successful attacks impractical, as users would likely notice the manipulation for higher ϵ values and the degradation in image quality reaches unacceptable levels.

Inspecting the impact of AT on cosine distances, before and after CW attacks, in Figure 7.4, we observe a reassuring pattern. The majority of the points are very close to the plot's identity or have only been reduced by a constant amount, indicating that AT can indeed be considered an efficient defense mechanism for our use-case.

	Maximal Perturbation				
Attack	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.3$
FGSM	0.01	0.03	0.02	0.02	0.00
PGD-8	0.01	0.07	0.42	3.18	6.91
PGD-16	0.01	0.07	0.48	4.33	12.04
PGD-32	0.01	0.07	0.48	5.12	15.43
PGD-64	0.01	0.07	0.50	5.45	16.97
PGD-128	0.01	0.07	0.53	5.61	17.68
CW-1000	0.00	0.30	1.80	23.10	48.00

Table 7.2: Attack success rates for $rank_{min} = 3$, targeting an adversarially trained model, calculated over 10,000 random article tuples (1,000 in the case of CW) for all evaluated attacks and various ϵ values.

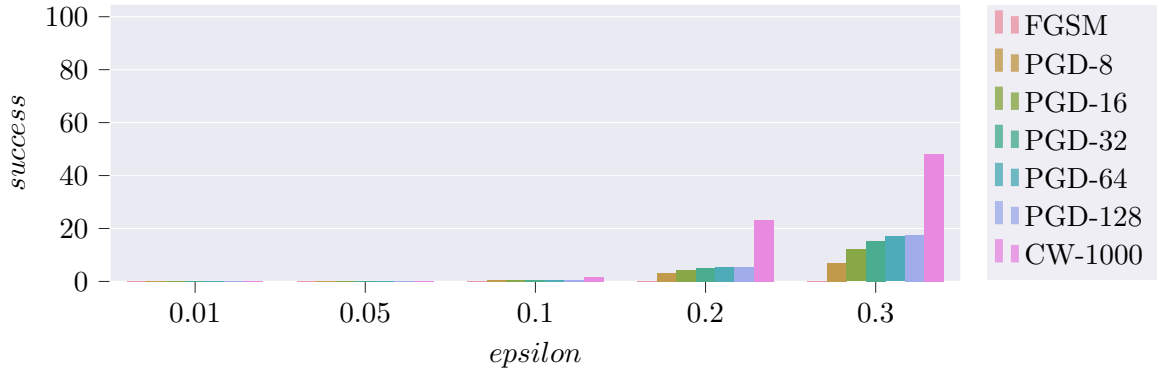


Figure 7.3: Attack success rates for $rank_{min} = 3$, targeting an adversarially trained model, calculated over 10,000 random article tuples (1,000 in the case of CW) for all evaluated attacks and various ϵ values.

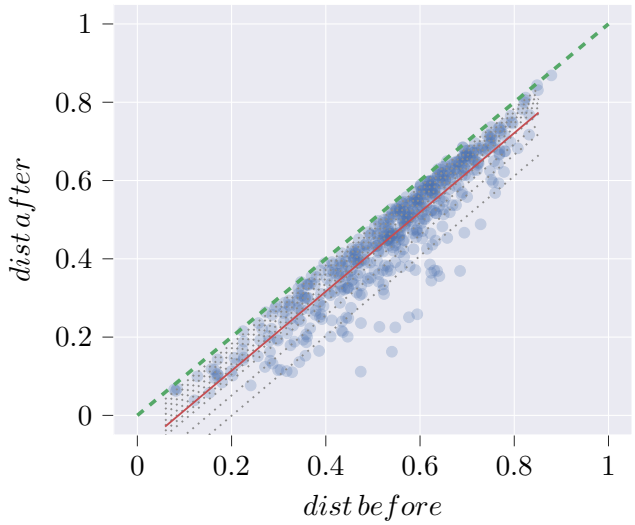


Figure 7.4: Quantile regression plot of 512 sampled cosine distances between target and attack article, before and after performing CW-1000 attacks targeting an adversarially trained model, using $\epsilon = 0.05$.

7.2 Curriculum Adversarial Training

Another defense from the class of AT defenses we evaluated is called curriculum adversarial training (CAT) and was first published by Cai et al., 2018. Using a curriculum of adversarial examples generated by attacks with a wide range of strengths, this approach is supposed to increase accuracy on clean and adversarial inputs for more complex tasks. We implemented and evaluated this approach, including its batch mixing optimization by training a model using PGD attacks with iterations up to $k = 8$, restricting l_∞ perturbations to $\epsilon = 0.03$. The evaluation results for this classifier can be seen in 7.3.

Category	Curriculum	Regular	Δ
Clean Accuracy	62.29	68.25	− 5.96
Adversarial Accuracy	27.45	0.02	+ 27.43

(a) Category classification results.

Texture	Curriculum	Regular	Δ
Clean Top-1 Precision	39.57	43.68	− 4.11
Adversarial Top-1 Precision	36.04	19.51	+ 16.53

(b) Texture classification results.

Table 7.3: Comparison of our evaluation results on a clean and adversarial test set for a classifier trained using curriculum adversarial training and a regular classifier. The adversarial test set was generated using the PGD-8 with $\epsilon = 0.03$

Looking at the results achieved by CAT, we observe a significant performance increase on clean data compared to traditional adversarial learning. However, this increase in performance on clean data comes at the cost of decreased robustness against adversarial examples. This lack of robustness becomes visible when we try to attack this model using our targeted item-to-item attack on our example attack tuple, as shown in Figure 7.5.

Figure 7.5: A recommendation result of our CAT model after a targeted attack. The adversarial example generated using the CW method for $\epsilon = 0.2$, which we injected into the product catalog, ranks first among the target's neighbors.

	Maximal Perturbation				
Attack	$\epsilon = 0.01$	$\epsilon = 0.05$	$\epsilon = 0.1$	$\epsilon = 0.2$	$\epsilon = 0.3$
FGSM	0.02	0.00	0.01	0.00	0.00
PGD-8	0.05	0.32	1.45	5.19	17.93
PGD-16	0.36	0.75	3.75	13.13	23.57
PGD-32	1.08	2.26	8.01	24.92	42.13
PGD-64	2.13	7.53	15.78	39.95	57.83
PGD-128	2.96	14.86	28.63	55.52	71.71
CW-1000	3.20	32.80	81.80	97.40	99.50

Table 7.4: Attack success rates for $rank_{min} = 3$ calculated over 10,000 random article tuples (1,000 in the case of CW) for all evaluated attacks and various ϵ values.

Examining the results of our empirical evaluation over a broader set of article tuples in Table 7.4 and Figure 7.6, our first impression is confirmed. Within a realistic perturbation budget of $\epsilon \leq 0.05$, the worst-case success probability reaches 32.80%, which we achieved using the CW method with 1,000 iterations. As the ϵ budget rises, the success rates reach levels well above 90% topping off at a 99.50% success rate for CW and $\epsilon = 0.3$.

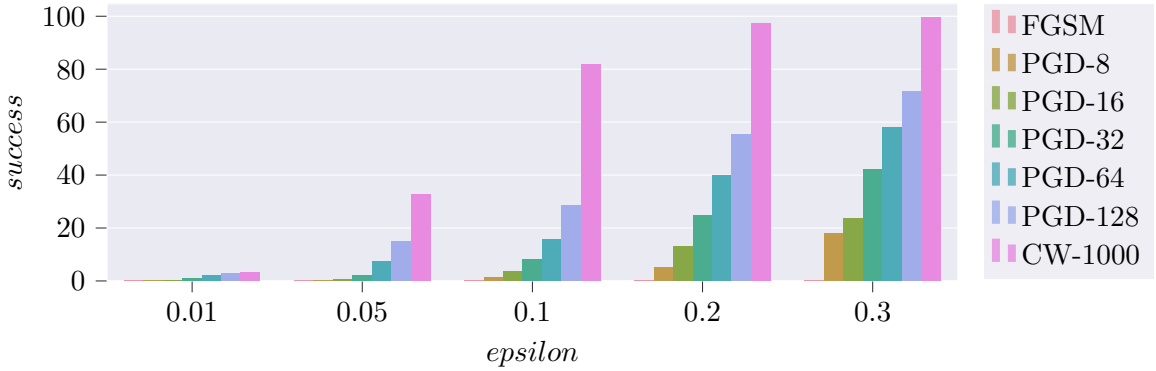


Figure 7.6: Attack success rates for $rank_{min} = 3$ calculated over 10,000 random article tuples (1,000 in the case of CW) for all evaluated attacks and various ϵ values.

Inspecting the impact of CAT on cosine distances, before and after CW attacks, in Figure 7.7, we observe a mediocre result. All points are located well below the plot's identity, but the reduction in distances is not bound to a constant like it was for traditional AT. Overall we can conclude that CAT achieved a balance between accuracy and robustness somewhere between regular training and traditional AT.

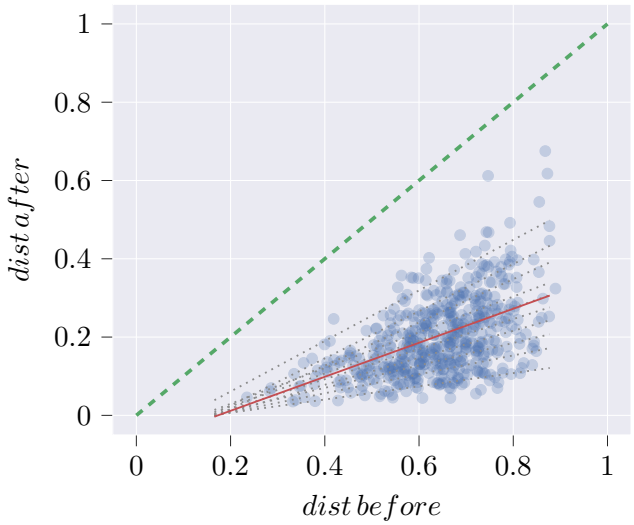


Figure 7.7: Quantile regression plot of 512 sampled cosine distances between target and attack article, before and after performing CW-1000 attacks, using $\epsilon = 0.05$.

7.3 Comparison

	Attack		
Defense	FGSM	PGD-128	CW-1000
Unsecured	0.07	98.32	99.70
Adversarial Training	0.03	0.07	0.30
Curriculum Adversarial Training	0.00	14.89	32.80

Table 7.5: Attack success rates for $rank_{min} = 3$ calculated over 10,000 random article tuples (1,000 in the case of CW) for all evaluated models and $\epsilon = 0.05$.

Finally, we want to compare our achieved results between the two evaluated defense methods. Table 7.5 shows the robustness of our trained models against the targeted item-to-item attacks, performed in Chapter 6. We observed that using traditional AT, we were able to significantly reduce the success rate for realistic perturbation budgets of $\epsilon \leq 0.05$. This result demonstrates that using adversarial examples for the training of a CNN feature extractor can effectively increase the robustness of the model against adversarial attacks targeting the hidden feature space. Although CAT also achieved to increase robustness in comparison to an undefended model, the increase is far lower than in the case of traditional AT. We should note that all measured robustness metrics only indicate robustness against the type of attacks and ϵ-boundaries we tested in our experiments and are no proof for universal robustness against other unknown attacks.

8 Conclusion

In this thesis, our goal was to investigate the vulnerability of a visual content-based recommendation system (RS) using a deep neural network (DNN) against adversarial attacks and to evaluate possible defense mechanisms. As a first step, we developed a new type of targeted item-to-item attack using state-of-the-art white-box methods and observed their effectiveness in compromising the integrity of the attacked RS. In the next step, we tested two defense mechanisms utilizing adversarial training (AT) and were able to show that AT had a significant positive impact on the robustness of our RS against our performed attacks.

Although our experiments demonstrated a strong robustness of adversarially trained content-based RSs against our evaluated white-box attacks, it is unclear if and how far these results generalize for black-box or future unknown attacks. Also, the effect of similar attacks and defenses on hybrid RSs using DNNs remains to be explored. Additionally, the trade-off in recommendation quality and robustness caused by AT remains to be quantified, possibly by conducting user-surveys or A/B testing.

Overall, our findings have once again demonstrated the inherent vulnerability of DNNs, but have also given us hope that adversarially robust RS models using DNNs might be within current reach.

List of Figures

List of Tables

List of Algorithms

Bibliography

Adomavicius, G., & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, *17*(6), 734–749. Retrieved August 24, 2020, from http://pages.stern.nyu.edu/~atuzhili/pdf/TKDE-Paper-as-Printed.pdf (cit. on pp. 18, 19)

Athalye, A., Carlini, N., & Wagner, D. (2018). Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1802.00420.pdf (cit. on pp. 24, 41, 46)

Bobadilla, J., Ortega, F., Hernando, A., & Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems*, *46*, 109–132. Retrieved August 24, 2020, from http://irntez.ir/wp-content/uploads/2016/12/sciencedirec.pdf (cit. on pp. 18–20)

Bracher, C., Heinz, S., & Vollgraf, R. (2016). Fashion dna: Merging content and sales data for recommendation and article mapping. *arXiv preprint arXiv:1609.02489*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1609.02489.pdf (cit. on p. 21)

Cai, Q.-Z., Du, M., Liu, C., & Song, D. (2018). Curriculum adversarial training. *arXiv preprint arXiv:1805.04807*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1805.04807.pdf (cit. on pp. 26, 49)

Carlini, N. (2019). *A Complete List of All Adversarial Example Papers*. Retrieved August 24, 2020, from https://nicholas.carlini.com/writing/2019/all-adversarial-example-papers.html (cit. on p. 6)

Carlini, N., & Wagner, D. (2017). Towards evaluating the robustness of neural networks, In *2017 ieee symposium on security and privacy (sp)*. IEEE. Retrieved August 24, 2020, from https://arxiv.org/pdf/1608.04644.pdf. (Cit. on pp. 22–24, 36)

Chen, P.-Y., Wu, S.-y., & Yoon, J. (2004). The impact of online recommendations and consumer feedback on sales. *ICIS 2004 Proceedings*, 58. Retrieved August 24, 2020, from https://aisel.aisnet.org/cgi/viewcontent.cgi?article=1146&context=icis2004 (cit. on p. 6)

Chen, T., Hong, L., Shi, Y., & Sun, Y. (2017). Joint text embedding for personalized content-based recommendation. *arXiv preprint arXiv:1706.01084*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1706.01084.pdf (cit. on p. 20)

Cheng, H.-T., Koc, L., Harmsen, J., Shaked, T., Chandra, T., Aradhye, H., Anderson, G., Corrado, G., Chai, W., Ispir, M., Et al. (2016). Wide & deep learning for recommender systems, In *Proceedings of the 1st workshop on deep learning for recommender systems*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1703.04247.pdf. (Cit. on p. 20)

Chirita, P.-A., Nejdl, W., & Zamfir, C. (2005). Preventing shilling attacks in online recommender systems, In *Proceedings of the 7th annual acm international workshop on web information and data management*. Retrieved August 24, 2020, from https://www.researchgate.net/profile/Cristian_Zamfir/publication/220759092_Preventing_shilling_attacks_in_online_recommender_systems/links/00b49536abca48b9cf000000.pdf. (Cit. on p. 7)

Christakopoulou, K., & Banerjee, A. (2019). Adversarial attacks on an oblivious recommender, 322–330. Retrieved August 24, 2020, from https://www-users.cs.umn.edu/~baner029/papers/19/adv_attack.pdf (cit. on p. 7)

Dalvi, N., Domingos, P., Sanghai, S., & Verma, D. (2004). Adversarial classification, In *Proceedings of the tenth acm sigkdd international conference on knowledge discovery and data mining*. Retrieved August 24, 2020, from https://dl.acm.org/doi/pdf/10.1145/1014052.1014066. (Cit. on pp. 6, 21)

Desrosiers, C., & Karypis, G. (2011). A comprehensive survey of neighborhood-based recommendation methods. In *Recommender systems handbook* (pp. 107–144). Springer. Retrieved August 24, 2020, from http://www.etsmtl.ca/getattachment/Professeurs/cdesrosiers/Publications/RecHB-SurveyOfNeighborhoodBasedRecommendation.pdf. (Cit. on p. 34)

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1810.04805.pdf (cit. on p. 21)

Di Noia, T., Malitesta, D., & Merra, F. A. (2020). Taamr: Targeted adversarial attack against multimedia recommender systems. Retrieved August 24, 2020, from https://arxiv.org/pdf/1706.01084.pdf. (Cit. on p. 7)

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(Jul), 2121–2159. Retrieved August 24, 2020, from http://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf (cit. on p. 11)

Fan, J., Keim, D. A., Gao, Y., Luo, H., & Li, Z. (2008). Justclick: Personalized image recommendation via exploratory search from large-scale flickr images. *IEEE Transactions on Circuits and Systems for Video Technology*, *19*(2), 273–288. Retrieved August 24, 2020, from https://www.researchgate.net/profile/Jianping_Fan2/publication/224358167_JustClick_Personalized_Image_Recommendation_via_Exploratory_Search_From_Large-Scale_Flickr_Images/links/0c96052532f8619ae1000000/JustClick-Personalized-Image-Recommendation-via-Exploratory-Search-From-Large-Scale-Flickr-Images.pdf (cit. on p. 21)

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. Retrieved August 24, 2020, from http://www.deeplearningbook.org. (Cit. on p. 13)

Goodfellow, I. J., Shlens, J., & Szegedy, C. (2014). Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1412.6572.pdf) (cit. on pp. 21, 22)

Grosse, K., Papernot, N., Manoharan, P., Backes, M., & McDaniel, P. (2016). Adversarial perturbations against deep neural networks for malware classification. *arXiv preprint arXiv:1606.04435*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1606.04435.pdf (cit. on p. 21)

Guo, H., Tang, R., Ye, Y., Li, Z., & He, X. (2017). Deepfm: A factorization-machine based neural network for ctr prediction. *arXiv preprint arXiv:1703.04247*. Retrieved August 24, 2020, from https://www.ijcai.org/Proceedings/2017/0239.pdf (cit. on p. 20)

Guzella, T. S., & Caminhas, W. M. (2009). A review of machine learning approaches to spam filtering. *Expert Systems with Applications*, *36*(7), 10206–10222. Retrieved August 24, 2020, from http://www.academia.edu/download/43906061/A_review_of_machine_learning_approaches_20160319-23716-gvrql2.pdf (cit. on p. 21)

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction, second edition*. Springer New York. Retrieved August 24, 2020, from https://web.stanford.edu/~hastie/Papers/ESLII.pdf. (Cit. on p. 16)

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition, In *Proceedings of the ieee conference on computer vision and pattern recognition*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1512.03385.pdf. (Cit. on pp. 15, 21)

He, R., & McAuley, J. (2016). Vbpr: Visual bayesian personalized ranking from implicit feedback, In *Thirtieth aaai conference on artificial intelligence*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1510.01784.pdf. (Cit. on p. 21)

He, X., Liao, L., Zhang, H., Nie, L., Hu, X., & Chua, T.-S. (2017). Neural collaborative filtering, In *Proceedings of the 26th international conference on world wide web*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1708.05031.pdf. (Cit. on p. 20)

He, X., Zhang, H., Kan, M.-Y., & Chua, T.-S. (2016). Fast matrix factorization for online recommendation with implicit feedback, In *Proceedings of the 39th international acm sigir conference on research and development in information retrieval*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1708.05024.pdf. (Cit. on p. 20)

Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T. N., Et al. (2012). Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*, *29*(6), 82–97. Retrieved August 24, 2020, from https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/HintonDengYuEtAl-SPM2012.pdf (cit. on p. 21)

Hinton, G., Srivastava, N., & Swersky, K. (2012). Lecture 6a overview of mini–batch gradient descent. Retrieved August 24, 2020, from https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. (Cit. on p. 11)

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359–366. Retrieved August 24, 2020, from https://www.cs.cmu.edu/~epxing/Class/10715/reading/Kornick_et_al.pdf (cit. on p. 12)

Ilyas, A., Engstrom, L., Athalye, A., & Lin, J. (2018). Black-box adversarial attacks with limited queries and information. *arXiv preprint arXiv:1804.08598*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1804.08598.pdf (cit. on p. 24)

Ilyas, A., Santurkar, S., Tsipras, D., Engstrom, L., Tran, B., & Madry, A. (2019). Adversarial examples are not bugs, they are features, In *Advances in neural information processing systems*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1905.02175.pdf. (Cit. on p. 21)

ISO Central Secretary. (2018). *Information security management systems — overview and vocabulary* (Standard ISO/IEC TR 27000:2018). International Organization for Standardization. Geneva, CH. Retrieved August 24, 2020, from https://www.iso.org/obp/ui/#iso:std:iso-iec:27000:en. (Cit. on p. 27)

Jain, T., Lennan, C., John, Z., & Tran, D. (2019). Imagededup. Retrieved August 24, 2020, from https://github.com/idealo/imagededup. (Cit. on p. 30)

Kang, W.-C., Fang, C., Wang, Z., & McAuley, J. (2017). Visually-aware fashion recommendation and design with generative image models, In *2017 ieee international conference on data mining (icdm)*. IEEE. Retrieved August 24, 2020, from https://arxiv.org/pdf/1711.02231.pdf. (Cit. on p. 21)

Karpathy, A. (2016). *Connecting images and natural language* (Doctoral dissertation). Stanford University. Retrieved August 24, 2020, from https://cs.stanford.edu/people/karpathy/main.pdf. (Cit. on pp. 8, 14)

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, *abs/1412.6980*arXiv 1412.6980. Retrieved August 24, 2020, from https://arxiv.org/pdf/1412.6980.pdf (cit. on pp. 11, 24, 33)

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks, In *Advances in neural information processing systems*. Retrieved August 24, 2020, from http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. (Cit. on pp. 15, 21)

Kurakin, A., Goodfellow, I., & Bengio, S. (2016a). Adversarial examples in the physical world. *arXiv preprint arXiv:1607.02533*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1607.02533.pdf (cit. on pp. 21, 23)

Kurakin, A., Goodfellow, I., & Bengio, S. (2016b). Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1611.01236 (cit. on p. 23)

Lam, S. K., & Riedl, J. (2004). Shilling recommender systems for fun and profit, In *Proceedings of the 13th international conference on world wide web*. Retrieved August 24, 2020, from http://vima01220.ethz.ch/CDstore/www2004/docs/1p393.pdf. (Cit. on p. 6)

Le, Q., & Mikolov, T. (2014). Distributed representations of sentences and documents, In *International conference on machine learning*. Retrieved August 24, 2020, from http://proceedings.mlr.press/v32/le14.pdf. (Cit. on p. 20)

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural computation*, *1*(4), 541–551. Retrieved August 24, 2020, from http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf (cit. on pp. 12, 15)

LeCun, Y., & Cortes, C. (2010). MNIST handwritten digit database. Retrieved August 24, 2020, from http://yann.lecun.com/exdb/mnist/ (cit. on p. 9)

Lin, J., Gan, C., & Han, S. (2019). Defensive quantization: When efficiency meets robustness. *arXiv preprint arXiv:1904.08444*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1904.08444.pdf (cit. on p. 24)

Liu, Z., Luo, P., Qiu, S., Wang, X., & Tang, X. (2016). Deepfashion: Powering robust clothes recognition and retrieval with rich annotations, In *Proceedings of ieee conference on computer vision and pattern recognition (cvpr)*. Retrieved August 24, 2020, from https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Liu_DeepFashion_Powering_Robust_CVPR_2016_paper.pdf. (Cit. on p. 30)

Ma, N., Zhang, X., Zheng, H.-T., & Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient cnn architecture design, In *Proceedings of the european conference on computer vision (eccv)*. Retrieved August 24, 2020, from https://openaccess.thecvf.com/content_ECCV_2018/papers/Ningning_Light-weight_CNN_Architecture_ECCV_2018_paper.pdf. (Cit. on p. 33)

Madry, A., Makelov, A., Schmidt, L., Tsipras, D., & Vladu, A. (2017). Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1706.06083) (cit. on pp. 21, 25, 46)

Mądry, A., & Schmidt, L. (2018). A brief introduction to adversarial examples. Retrieved August 24, 2020, from https://gradientscience.org/intro_adversarial/ (cit. on p. 22)

Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1603.09320.pdf (cit. on p. 34)

McAuley, J., Targett, C., Shi, Q., & Van Den Hengel, A. (2015). Image-based recommendations on styles and substitutes, In *Proceedings of the 38th international acm sigir conference on research and development in information retrieval*. Retrieved August 24, 2020, from https://dl.acm.org/doi/pdf/10.1145/2766462.2767755. (Cit. on p. 21)

Morgan, N., & Bourlard, H. (1990). Continuous speech recognition using multilayer perceptrons with hidden markov models, In *Acoustics, speech, and signal processing, 1990. icassp-90., 1990 international conference on*. IEEE. (Cit. on p. 12).

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines, In *Proceedings of the 27th international conference on machine learning (icml-10)*. Retrieved August 24, 2020, from https://www.cs.toronto.edu/~hinton/absps/reluICML.pdf. (Cit. on p. 17)

Papernot, N., McDaniel, P., Wu, X., Jha, S., & Swami, A. (2016). Distillation as a defense to adversarial perturbations against deep neural networks, In *2016 ieee symposium on security and privacy (sp)*. IEEE. Retrieved August 24, 2020, from https://arxiv.org/pdf/1511.04508.pdf. (Cit. on p. 24)

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Et al. (2019). Pytorch: An imperative style, high-performance deep learning library, In *Advances in neural information processing systems*. (Cit. on p. 18).

Pazzani, M. J., & Billsus, D. (2007). Content-based recommendation systems. In *The adaptive web* (pp. 325–341). Springer. (Cit. on p. 20).

Rojas, R. (2013). *Neural networks: A systematic introduction*. Springer Science & Business Media. Retrieved August 24, 2020, from https://page.mi.fu-berlin.de/rojas/neural/neuron.pdf. (Cit. on p. 16)

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Et al. (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, *115*(3), 211–252. Retrieved August 24, 2020, from https://arxiv.org/pdf/1409.0575) (cit. on p. 9)

Sahs, J., & Khan, L. (2012). A machine learning approach to android malware detection, In *2012 european intelligence and security informatics conference*. IEEE. Retrieved August 24, 2020, from http://csis.pace.edu/ctappert/dps/2012EISIC/data/4782a141.pdf. (Cit. on p. 21)

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L.-C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks, In *Proceedings of the ieee conference on computer vision and pattern recognition*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1801.04381.pdf. (Cit. on p. 33)

Scarfone, K., Jansen, W., & Tracy, M. (2008). Guide to general server security. *NIST Special Publication*, *800*(s 123). Retrieved August 24, 2020, from https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-123.pdf (cit. on p. 29)

Shamir, O., & Zhang, T. (2013). Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes, In *International conference on machine learning*. Retrieved August 24, 2020, from http://proceedings.mlr.press/v28/shamir13.pdf. (Cit. on p. 10)

Shankar, D., Narumanchi, S., Ananya, H., Kompalli, P., & Chaudhury, K. (2017). Deep learning based large scale visual recommendation and search for e-commerce. *arXiv preprint arXiv:1703.02344*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1703.02344.pdf (cit. on p. 21)

Shin, H.-C., Roth, H. R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D., & Summers, R. M. (2016). Deep convolutional neural networks for computer-aided detection: Cnn architectures, dataset characteristics and transfer learning. *IEEE transactions on medical imaging*, *35*(5), 1285–1298. Retrieved August 24, 2020, from https://arxiv.org/pdf/1602.03409.pdf (cit. on p. 33)

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1409.1556.pdf (cit. on p. 15)

Stanford. (2020). Cs231n: Convolutional neural networks for visual recognition. Retrieved August 24, 2020, from https://cs231n.github.io/convolutional-networks/ (cit. on pp. 13, 15)

Su, J.-H., Huang, W.-J., Philip, S. Y., & Tseng, V. S. (2010). Efficient relevance feedback for content-based image retrieval by mining user navigation patterns. *IEEE transactions on knowledge and data engineering*, *23*(3), 360–372. Retrieved August 24, 2020, from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.717.9745&rep=rep1&type=pdf (cit. on p. 21)

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions, In *Proceedings of the ieee conference on computer vision and pattern recognition*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1409.4842.pdf. (Cit. on p. 15)

Szegedy, C., Zaremba, W., Sutskever, I., Bruna, J., Erhan, D., Goodfellow, I., & Fergus, R. (2013). Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1312.6199.pdf (cit. on pp. 6, 21, 24, 46)

Tang, J., Du, X., He, X., Yuan, F., Tian, Q., & Chua, T.-S. (2019). Adversarial training towards robust multimedia recommender system. *IEEE Transactions on Knowledge and Data Engineering*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1809.07062.pdf (cit. on p. 7)

Tramèr, F., Kurakin, A., Papernot, N., Goodfellow, I., Boneh, D., & McDaniel, P. (2017). Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1705.07204.pdf) (cit. on p. 24)

Tsipras, D., Santurkar, S., Engstrom, L., Turner, A., & Madry, A. (2018). Robustness may be at odds with accuracy. *arXiv preprint arXiv:1805.12152*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1805.12152.pdf (cit. on p. 47)

Tuinhof, H., Pirker, C., & Haltmeier, M. (2018). Image-based fashion product recommendation with deep learning, In *International conference on machine learning, optimization, and data science*. Springer. Retrieved August 24, 2020, from https://arxiv.org/pdf/1805.08694.pdf. (Cit. on pp. 21, 30, 32, 34)

Xing, X., Meng, W., Doozan, D., Snoeren, A. C., Feamster, N., & Lee, W. (2013). Take this personally: Pollution attacks on personalized services, In *22nd usenix security symposium (usenix security 13)*. Retrieved August 24, 2020, from https://www.usenix.org/system/files/conference/usenixsecurity13/sec13-paper_xing.pdf. (Cit. on p. 6)

Yang, G., Gong, N. Z., & Cai, Y. (2017). Fake co-visitation injection attacks to recommender systems., In *Ndss*. Retrieved August 24, 2020, from http://people.duke.edu/~zg70/papers/ndss17-attackRS.pdf. (Cit. on p. 6)

Yin, C., Zhu, Y., Fei, J., & He, X. (2017). A deep learning approach for intrusion detection using recurrent neural networks. *Ieee Access*, *5*, 21954–21961. Retrieved August 24, 2020, from https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8066291 (cit. on p. 21)

Zhang, Y., & Yang, Q. (2017). A survey on multi-task learning. *arXiv preprint arXiv:1707.08114*. Retrieved August 24, 2020, from https://arxiv.org/pdf/1707.08114.pdf (cit. on p. 32)

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

This thesis was not previously presented to another examination board and has not been published.

City, Date

Philipp Normann