ABSTRACT

Title of Dissertation: ADVERSARIAL MACHINE LEARNING IN THE WILD

Parsa Saadatpanah
Doctor of Philosophy, 2021

Dissertation Directed by: Professor Tom Goldstein
Computer Science Department

Deep neural networks are making their way into our everyday lives at an increasing rate. While the adoption of these models has greatly improved our everyday lives, it has also opened the door to new vulnerabilities in real-world systems. More specifically, in the scope of this work we are interested in one class of vulnerabilities: adversarial attacks. Given the high importance and the sensitivity of some of the tasks these models are responsible for, it is crucial to study such vulnerabilities in real-world systems. In this work, we look at examples of deep neural network-based real-world systems, vulnerabilities of such systems, and approaches for making such systems more robust.

First, we study an example of leveraging a deep neural network in a business-critical real-world system. We discuss how deep neural networks improve the quality of smart voice assistants. More specifically, we introduce how collaborative filtering models can automatically detect and resolve the errors of a voice assistant. We then discuss the success of this approach in improving the quality of a real-world voice assistant.

Second, we demonstrate a proof of concept for an adversarial attack against content-

based recommendation systems which are commonly used in real-world settings. We discuss how malicious actors can add unnoticeable perturbations to the content they upload to the website to achieve their preferred outcomes. We also show how adversarial training can render such attacks useless.

Third, we discuss another example of how adversarial attacks can be leveraged to manipulate a real-world system. We study how adversarial attacks can successfully manipulate YouTube's copyright detection model and the financial implications of this vulnerability. In particular, we show how adversarial examples created for a copyright detection model that we implemented transfer to another black-box model.

Finally, we study the problem of transfer learning in an adversarially robust setting. We discuss how robust models contain robust feature extractors and how we can leverage them to train new classifiers that preserve the robustness of the original model. We then study the case of fine-tuning in the target domain while preserving the robustness. We show the success of our proposed solutions in preserving the robustness in the target domain.

ADVERSARIAL MACHINE LEARNING IN THE WILD

by

Parsa Saadatpanah

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2021

Dedication

To my beloved wife, loving parents, and supportive brother.

Acknowledgments

First and foremost, I owe my gratitude to my advisor Professor Tom Goldstein for guiding me through my Ph.D. journey. Above everything else, I appreciate that he patiently allowed me to follow my interests and pursue my own goals. I am also incredibly thankful for his continuous support, constructive feedback, and great ideas. Whenever I felt stuck, he was there to help me find my next step and make progress. Not only I have learned a lot in his class as a student, but also I learned how to be a good researcher and think like a researcher from him. Tom made this journey feel exciting and fun rather than like the massive endeavor it is.

Many Thanks to Wojciech Czaja, John Dickerson, Tom Goldstein, Furong Huang, and Christopher Metzler for devoting their time and serving on my thesis committee, and for their invaluable advice and suggestions.

I am thankful to all my fellow collaborators and co-authors who made all of this work possible. It was a privilege to be able to work with so many bright researchers throughout my Ph.D. journey.

I would also like to express my heartfelt gratitude for the unconditional support of my friends throughout this journey, Saba Ahmadi, Soheil Behnezhad, Sina Dehghani, Mahsa Derakhashan, Soheil Ehsani, Hossein Esfandiari, Alireza Farhadi, Amin Ghiasi, Mahyar Najibi, Kiana Roshanzamir, Hamed Saleh, Saeed Seddighin, Ali Shafahi, and

Hadi Yami.

Last but not least, I owe my deepest thanks to my family. First and foremost, my beloved wife Madeline for being there for me no matter what and always believing in me. And my loving parents Mahboubeh and Alireza, and my brother Pooya, without whom this journey would not have been possible.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

In recent years, Deep Neural Networks (DNNs) have been making their way into our everyday lives at an increasing rate. Voice assistants, autonomous cars, smart home security cameras, and recommendation systems are a few examples of widely used real-world systems that are highly dependent on DNNs for their functionality. In many cases, such systems are considered business-critical, and their malfunction can result in significant financial loss, legal loss, or even human casualties. For example, a malfunction in the object detection model of an autonomous car can result in the car not detecting a pedestrian is on the road. Or, a malfunction in the Automatic Speech Recognition model or the Natural Language Processing model of a Voice Assistant can result in a message (possibly containing sensitive information) being sent to the wrong recipient. Furthermore, the business-critical nature of these systems creates a real incentive for malicious actors to try to attack these systems. For example, a business rival might try to decrease the revenue of an online store by causing poor product recommendations.

The rapid adoption of DNNs can be attributed to their significantly high performance on many benchmarks and in real-world tasks. By automatically extracting meaningful features out of large, complex datasets, these models have become the obvious choice for many real-world use cases. However, there are situations in which these models perform very poorly. The most common cause of a model's performance degradation in real-world settings is distribution shifts. DNNs are very sensitive to the distribution of the data they have been trained on, and the general working assumption for them is that the test data is from the same distribution as the training data. However, in a real-world setting, the distribution of the data is constantly changing. For example, the distribution of what people buy on an online store during December is very different than what they would buy in January. So a recommendation system trained on the data from December, would not perform well on January 1st. However, distribution shifts are not the only scenario in which models perform poorly in real-world settings. Adversarial attacks can also cause these real-world models to have a degraded performance.

1.1 Adversarial Attacks

Adversarial attacks refer to a family of attacks against machine models in which a malicious actor makes small changes to the input of the model to cause large changes to the output. For example, in the computer vision domain, an adversary can slightly change an image to cause it to be misclassified by the machine learning model while keeping the change unnoticeable to human eyes. These attacks have been demonstrated on some potentially sensitive systems, largely in an idealized academic context, and occasionally in the real-world [1, 2, 3, 4, 5, 6].

There are two major types of adversarial attacks: targeted and non-targeted. In a non-targeted attack, the adversary only tries to make sure that the output of the perturbed input is as different as possible from the true output. The optimization problem in equation 1.1

describes the non-targeted adversarial attacks. In this equation, δ refers to the perturbation, x is the input to the model, y is the true output, θ is the set of parameters that define the model, L is the loss function, and ϵ defines an upper bound for how big the perturbations can be.

$$\max_{\delta} L\left(x+\delta, y; \theta\right) \qquad s.t. \left\|\delta\right\|_p \leq \epsilon \tag{1.1}$$

On the other hand, in targeted attacks, the adversary tries to make the output of the model as similar as possible to a particular target. Similar to non-targeted attacks, we can formulate targeted adversarial attacks with an optimization problem as described in equation 1.2. Here, y_t refers to the target output for the attack.

$$\min_{\delta} L\left(x+\delta, y_t; \theta\right) \qquad s.t. \left\|\delta\right\|_p \leq \epsilon \tag{1.2}$$

1.2 Adversarial Robustness

A machine learning model is considered robust if it can maintain its accuracy against adversarial attacks. Although naturally trained DNNs often completely fail under adversarial inputs [7, 8], there are different approaches to train DNNs that are more robust to such attacks. The problem of training models that are robust to adversarial attacks has been studied for different types of adversarial attacks [9, 10]. Generally speaking, these approaches train the model by not just trying to minimize the loss function, but rather trying to minimize the loss function in the presence of deliberately chosen adversarial perturbations.

One of the most common approaches for achieving this goal is to use adversarial training, in which adversarial examples are crafted on the fly during network training and injected into the training set. While this approach greatly improves the robustness of DNNs against adversarial examples, it significantly increases the training time of the model. To create strong adversarial examples, we need to run multiple iterations of projected gradient descent for each training example where each iteration requires a forward and a backward pass. So training an adversarially robust model would need k-times more computation where k is the number of PGD iterations. The substantial difference in the training time of a robust model compared to a naturally trained model complicates the adaption of robust models in real-world settings.

Furthermore, it is known that a relatively large dataset is required to train a robust model [11]. The data-hungry nature of these robust models makes them a poor fit for settings where we don't have access to a large amount of data to train on. Specifically, in many real-world settings, the cost associated with collecting and labeling large volumes of data can be prohibitive. Therefore, for integrating robust models in real-world settings, we need an alternative approach that can work with small datasets.

1.3 Adversarial Attacks in Real-World Settings

Adversarial attacks have been widely studied in idealized academic contexts and on benchmark datasets. While there are a few works that look into the vulnerability of real-world systems such as self-driving cars [1], and high-frequency trading systems [12] to adversarial attacks, the problem of adversarial attacks against real-world systems is generally unexplored. Given the significant role that machine learning models play in real-world systems and the financial and legal consequences of a failure for these systems, the problem of building adversarially robust machine learning models for real-world use cases needs to be studied more extensively.

It can be argued that more research contributions on demonstrating the effectiveness of adversarial attacks against the real-world system can incentivize and justify the future work for building adversarially robust models. By demonstrating successful adversarial attacks against major real-world systems as a proof of concept, we can encourage the community to address the vulnerability of these systems to adversarial attacks. In particular, by targeting real-world systems that influence billions of dollars in revenue we can emphasize the urgency of this problem.

A copyright detection system is an example of a real-world system the heavily relies on machine learning models and is responsible for ensuring billions of dollars are revenue are not lost by content creators and copyright holders. In this setting, a malicious actor is incentivized to use an adversarial example to circumvent these systems to upload copyright-protected content and benefit from the ad or subscription revenue generated by that content.

A recommendation system is another example of the application of machine learning in a real-world system. Online stores heavily rely on recommendation systems to show the most relevant items to the users and maximize their revenue which, can be hundreds of billions of dollars. In this setting, a seller can leverage adversarial examples to force the system to recommend their product more frequently and to more users to increase their sales revenue.

1.4 outline

In Chapter 2, we discuss an example of a critical real-world system that heavily relies on a machine learning model. In particular, we address the problem of error detection and error resolution for smart voice assistants, and how a machine learning collaborative filtering model can be utilized to solve the problem. Despite all the advancements in Automated Speech Recognition (ASR) and Natural Language Processing (NLP) systems, voice assistants are still not perfectly accurate. Many times a voice assistant error either fails to do anything or does something unrelated in response to a voice command. In this chapter, we introduce a novel approach for error detection and resolving such errors by using a collaborative filtering model.

We then discuss a proof of concept for a real attack against a recommendation system in chapter 3. Specifically, we propose a new attack model that targets cold-start recommendations — i.e the situation where we don't have access to the historical user-item interaction data for a newly added item or a newly joined user. In this setting, we look at content-based methods where metadata and the content information of a new item are used for generating recommendations. In our proposed attack, a malicious actor can achieve their desired outcome by making small perturbations to the content information of a new item while making sure the perturbations are not noticeable by a human. We then discuss a defense mechanism against this attack. Particularly, we show that by using adversarial training we can render such attacks useless.

In chapter 4, we explore another adversarial attack against a real-world system. In this chapter, we discuss a novel mechanism for crafting adversarial examples that can evade copyright detection systems. By using a gradient-based attack we create adversarial music that is easily recognizable to a human while evading detection by a machine. The adversarial examples created by our approach have a very good transferability to black-box models. In particular, we show that the adversarial examples created by our approach can successfully fool proprietary industrial systems.

Finally, in chapter 5 we discuss a new approach for training machine learning models that are robust against adversarial attacks. In particular, we address the problem of training adversarially robust models when we have a limited amount of training data available to us, or the computational resources available to us at training time are limited. We start by observing that robust networks contain robust feature extractors. By training classifiers on top of these feature extractors, we produce new models that inherit the robustness of their parent networks. We then consider the case of "fine-tuning" a network by re-training in the target domain. By using this approach, it is possible to produce accurate and robust models with little data, and without the cost of adversarial training. Additionally, we can improve the generalization of adversarially trained models, while maintaining their robustness.

Chapter 2: Voice Assistant Error Resolution via Collaborative Filtering

2.1 Introduction

The rapid advancements of Machine Learning in recent years have drastically changed our everyday lives. As a part of this change, speech-driven smart agents (also known as Voice Assistants) such as Amazon Alexa, Google Assistant, or Apple Siri have become an integral part of our lives. The simplicity and convenience of interacting with voice assistants have made us accustomed to using our voice assistants for many everyday tasks such as controlling our smart homes, making phone calls, setting reminders, and making general inquiries. This convenience is a result of voice assistants' ability to understand our spoken language.

Any voice assistant consists of two main components: 1) an Automatic Speech Recognition (ASR) system, and 2) a Natural Language Processing (NLP) system. When a user directs a spoken command at a voice assistant, the recorded audio signal is sent to the ASR system to get the command transcription. Then, the transcription is sent to the NLP system to understand the user's command and then choose an action accordingly. Voice assistants owe their impressive accuracy and quality to recent advancements in both ASR and NLP. Novel ASR models [13, 14, 15, 16] have enabled the voice assistants to accurately transcribe the user commands in the noisiest environments. Furthermore, advancements in NLP and voice search in particular [17, 18, 19, 20, 21] have made it possible for the users to have the most natural speech-based interactions with their voice assistants.

Voice assistant errors: Despite all the advancements in ASR and NLP, voice assistants are still not perfectly accurate. Many of us have experienced hearing "Sorry, I didn't quite get that" from a voice assistant in response to a voice command we had issued. Or even worse, voice assistants sometimes misunderstand us and do something entirely unrelated. In this work, we define a voice assistant error as the voice assistant either failing to do anything or doing something unrelated in response to a voice command. While these errors are sometimes unavoidable because the user wants something beyond the system capabilities, many of the errors are in response to voice commands that are well within the capabilities of the voice assistant. Generally speaking, such voice assistant errors can be caused by one of the two following reasons: 1) the ASR system produces a wrong transcription from the audio signal, or 2) the NLP system is not able to correctly understand or respond to the voice command.

Voice assistant errors negatively impact the user experience and therefore need to be addressed. Previous works have approached this problem as a predictive task: given a command, predict the correct response. This requires a significant amount of labeled training data which would need to be collected by first identifying all voice assistant errors, and then finding the correct labels for them. Shokouhi et al. [22] use heuristics based on click-through logs to guess the correct transcription label (for ASR) whereas Rao et al. [18] use similar heuristics based on viewing patterns to guess the appropriate NLP response.

While prior work has focused on holistically replacing the existing voice assistant with a predictive ML model, this approach can sometimes be too disruptive in an existing real-world voice assistant product. Business considerations often warrant a more tactful approach — separating the errors out of a potentially very large number of user commands and then providing a resolution to fix each error. For the rest of this chapter, we will refer to the first part of this process as *Error Detection* and the second part as *Error Resolution*.

Error Detection is straightforward if the voice assistant simply did not understand the command and failed to take any action (resulting in a message such as "Sorry, I didn't quite get that"). These cases can be automatically extracted from voice command logs. However, it can be very challenging to detect cases where the voice assistant misunderstands a command and takes the wrong action. In this scenario, everything looks normal from the voice assistant's perspective, making it hard to distinguish from correctly handled cases. Even humans would require to spend a significant amount of effort to identify these errors. For example, if an annotator were to only look at the transcriptions and not to listen to the audio, she would think the transcription "CVS" (a pharmacy) is correct while the user had said "CBS" (a TV channel). Among the millions of voice commands issued by users every day, error detection becomes an impossible task for human annotators.

Error Resolution, on the other hand, requires knowledge of the system as well as the domain. Following the previous example, even if we knew that the response "CVS" was a voice assistant error, finding a resolution would require the knowledge that there is a TV channel called "CBS" which sounds similar to "CVS", making it a likely ASR error. Going through errors every day and making this judgment requires time and effort from skilled annotators.

In this work, we propose automated processes for error detection and error resolution, as well as an optional manual verification step. We use the “wisdom of the crowd” in an unsupervised fashion to solve both problems. We combine explicit signals (e.g., user repeating the same command in the same session) with a collaborative filtering model [23] trained directly on voice command logs to predict the likelihood of a voice assistant error. The model is also used to extract item-item similarities and suggest other commands as a resolution. To the best of our knowledge, this work is the first time collaborative filtering is being used to directly identify and resolve voice assistant errors. While utilizing voice command logs as a source of ground truth has been previously explored [18, 22], it has never been used in the context of collaborative filtering.

The main contributions of this work are the following:

- We introduce a novel approach for error detection in a voice assistant through the utilization of voice command logs.
- We introduce a novel approach for error resolution based on collaborative filtering. To the best of our knowledge, we are the first to use collaborative filtering for such a task or in the context of voice assistants and voice search.
- We demonstrate the effectiveness of our proposed approach by evaluating it against one of the most popular voice assistants used in the United States with tens of millions of users.

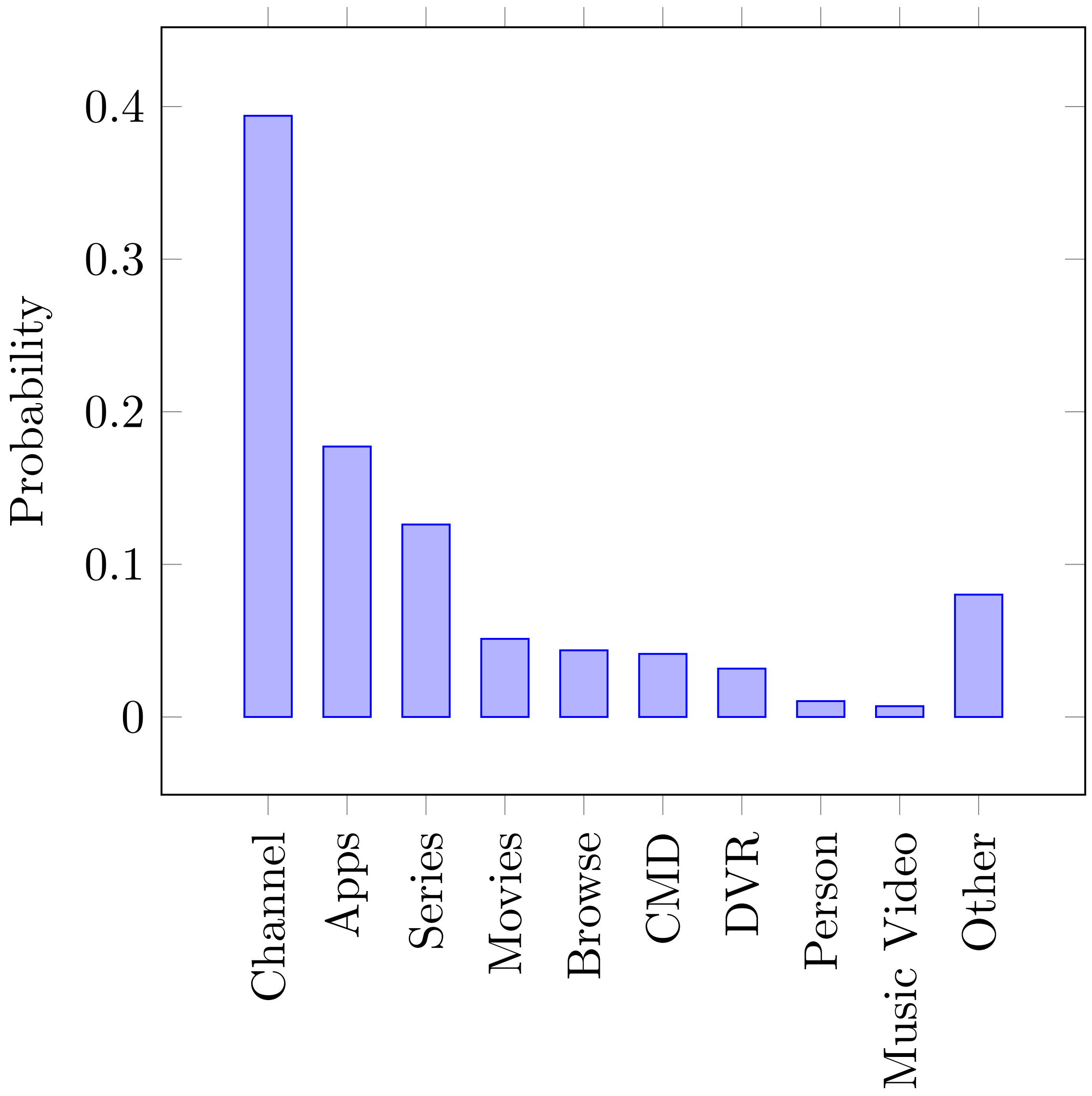


Figure 2.1: The long tail distribution of top families of action

2.2 Voice Assistant Platform

Our voice assistant platform has reinvented the way that customers can discover and access content on an entertainment platform. The voice remote delivers a simple and fast access to live, recorded, and on-demand entertainment, sports, and news content.

This voice remote allows the users to find what they are looking for with spoken language as input. The voice command issued by the user is first transcribed to text by an Automatic Speech Recognition (ASR) system and then is processed by a Natural Language Processing (NLP) system to map the user's voice command to an appropriate action (e.g. "tune to a TV channel") to be executed on TV.

Each year, several billion voice commands are received by this voice remote. The voice commands are mapped into more than a hundred different categories of user intents and executable actions (such as "tune to a TV channel", "open third-party applications", "find a movie/show title", "browse a collection of movies/shows by genre" etc.). A taxonomy for user intents to categorize different types of commands has been created and a combination of hand-crafted rules and machine-learning models are detecting the user intent for a given voice command [24].

Although users issue hundreds of thousands of unique voice commands, those commands have a very long tail distribution. The top 10 unique voice commands issued by the users consist of approximately 30% of the total voice command logs. This observation also holds across families of actions (intents). The top action ("tune to a TV channel") alone consists of almost 40% of the total voice command logs. Figure 2.1 shows the distribution of top intents.

Error detection and resolution of our entertainment platform voice assistant is a cumbersome task and with the very large amount of voice command logs that are generated every day it is impossible to manually verify the response of the ASR and NLP systems. In this work we share the techniques and data-driven approaches that we use for error detection and error resolution.

2.3 Error Detection

As described earlier, voice assistant errors can be caused by either the ASR system producing a wrong transcription from the audio signal or the NLP system not being able to understand the voice command or respond to it correctly. We categorize voice assistant errors into two types: 1) voice assistant's failure to take any action in response to voice command and 2) voice assistant executing an action unrelated to a voice command. In what follows we describe how we aim at detecting these errors.

Failure-based error detection: Error detection is straightforward if the voice assistant simply was not able to respond to a voice command (resulting in a message such as "Sorry, I didn't quite get that"). Whether ASR failed at transcribing the audio signal correctly or NLP failed at understanding and responding to a voice command, these errors can be easily extracted from the voice command logs.

Voice session-based error detection: Generally speaking, most user interactions with the voice assistant consist of issuing one voice command followed by a long period of silence. This happens because the users will get their desired content or outcome after issuing the first voice command and do not need to interact with the voice assistant any further. However, a user sometimes issues a sequence of back-to-back voice commands in a relatively short amount of time. This pattern of short bursts of multiple voice commands (aka voice session) appears across other voice assistant and voice search platforms as well [18, 21, 22]. The definition of a "voice session" varies but in this work we borrow the definition of Rao et al. [24] for a "voice session": a sequence of consecutive commands with a maximum gap of less than 45 seconds between commands. Figure 2.2 shows

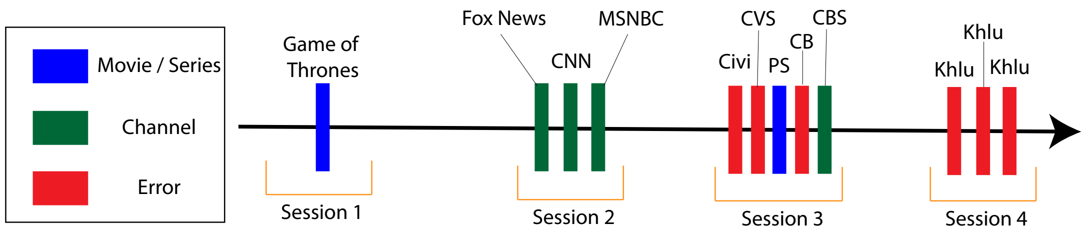


Figure 2.2: Four voice session examples.

several examples for a voice session. As shown in this figure, a voice session can either contain only a single voice command (session 1 - this is the case for most of the voice sessions) or it can contain multiple voice commands (sessions 2, 3, and 4). There are several cases in which a voice suggestion would have multiple voice commands. In some of the long sessions, the user may be simply exploring several channels, movies, and shows. Session 2 in figure 2.2 represents such a case where the user appears to be in an exploratory mood. Most importantly, a voice assistant error can result in a voice session where the user repeats the same or a similar voice command over and over within a short period. Sessions 3 and 4 in figure 2.2 represent two of such cases where the user appears to be having an unsatisfactory experience and is repeating the same voice command over and over in the hope of averting the voice assistant error.

Based on the observation that users tend to repeat their voice command in one voice session when they face an error, we can conclude that whether or not the same voice command gets repeated in a single voice session can be a strong signal for detecting the errors of the voice assistant. Therefore, we propose the following approach to automatically identify the problematic voice commands that are impacting a lot of voice sessions. For each voice command we compute these two values: 1) $\text{session}_r(c)$ = number of voice sessions in which the voice command c has been repeated more than once, and 2)

$\text{session}_t(c)$ = total number voice sessions in which the voice command c has appeared at least once. We then compute the following metric as a measure of how likely the voice command c is an error:

$$m(c) = \frac{\text{session}_r(c)}{\text{session}_t(c)} \tag{2.1}$$

This measure can then be used to identify the errors which we would have otherwise missed using the failure-based detection method described earlier. Given this measure, we can come up with a threshold θ for which we can say with high confidence that the voice commands that have $m(c) > \theta$ are errors. Furthermore, this measure will allow the QA engineers to focus their efforts on the voice commands that have a higher probability of being an error (the voice commands with higher $m(c)$), rather than randomly looking for errors among millions of voice commands.

Although the techniques described in this section will allow us to identify the voice assistant errors, we still have no way of knowing what is the correct resolution for those errors. For example, we need a mechanism to identify that the correct resolution for the voice command "CVS" is "CBS" (refer to the examples in figure 2.2). In the following section, we describe how collaborative filtering can be applied to voice assistant usage logs to identify the correct resolution of the errors in an unsupervised fashion (without the need for any annotated data).

2.4 Error Resolution via Collaborative Filtering

Collaborative filtering models are frequently used for training and building recommender systems in various domains. In short, a recommendation system's task is to provide a ranked list of the most relevant items (products, movies, etc.) to a user based on the user's past interactions with other items. The past interactions of a user can be either in the form of ratings given to different items by the user (explicit feedback), or it can be just the history of all the items the user has consumed (implicit feedback). Implicit feedback recommendation can be viewed as predicting the probability of how relevant an item is to a user, given the history of all items the user has consumed. In other words, in implicit feedback recommendation we are interested in predicting the value of $P(c|H_u)$ where c is an item, u is a user, and $H_u = [c_{u,1}, c_{u,2}, \cdots, c_{u,k}]$ is the history of all the items user u has previously consumed. This setting makes implicit feedback collaborative filtering models the perfect candidate for our case since we don't want to use any annotated data and we only have access to the list of voice commands each user has issued.

In this work, we train a collaborative filtering model on the voice command logs of a voice assistant. Training such a model will allow us to estimate how interchangeable two voice commands are for the purpose of computing the probability $P(c|H_u)$. This similarity notion is referred to as item-item similarity in the recommendation systems domain and using it allows us to find a resolution for an error by finding the most suitable replacement for that error. To train this model, we treat each voice command as a single item (e.g. "show me NBC" is treated as one item), and the voice assistant usage logs (all the voice commands issued by a given user u) are treated as the history H_u. Note that in this work we do not train the model at the word level, but rather at the voice command level. In other words, the voice commands "NBC" and "show me NBC" are treated as completely different items and we do not use the information about what words each command contains at all. Furthermore, it's worth mentioning that for training such a collaborative filtering model we only use the usage logs of the voice assistant and we do not need any form of annotated data.

Recently, a number of novel neural network-based collaborative filtering methods are emerging [25, 26, 27, 28]. We believe the model we propose in this work is most similar to [27] with some modifications to the training process and loss function.

2.4.1 Collaborative Filtering Model

The first step is to transform every voice command in H_u to their corresponding latent vector representation. To do so, we use matrix $Y_{|C|\times d}$ where C is the set of all voice commands issued by all users and d is the embedding dimension. In order to transform a voice command c to its latent vector representation, we will multiply its one-hot vector representation e_c (a $|C|$ dimensional vector that has value 1 in the c-th dimension and 0 everywhere else) by matrix Y. We apply this transformation to all voice commands in list H_u to get a list of d-dimensional vector representations $[e_{c_{u,1}}Y, e_{c_{u,2}}Y, \cdots, e_{c_{u,k}}Y]$. As described in equation 2.2, we define the user's vector representation X_u as the average of vector representations of voice commands in H_u:

$$X_u = \frac{1}{|H_u|}\sum_{c\in H_u}\text{Vector}(c) = \frac{1}{|H_u|}\sum_{c\in H_u}(e_cY) = \left(\frac{1}{|H_u|}\sum_{c\in H_u}e_c\right)Y \quad (2.2)$$

After Computing the user vector, we need to project it into the decision space. To do so, the user vector is passed through multiple layers of feed-forward fully connected neural networks. It has been shown that doing so helps with the recommendation accuracy of the model. We have also noticed a slight improvement in the embedding quality because of the projection step. We refer to the output of the last feed-forward layer as the projected user vector X_u^p. After computing this vector, we compute a score for all voice commands in the set C based on how similar they are to X_u^p. To do so, we multiply X_u^p with another matrix $Y'_{|C|\times d}$. Similar to Y, each row of this matrix is a d dimensional vector representation for a voice command. Using the Softmax function, we define the probability of recommending voice command c to the user as

$$P(c|H_u) = \frac{e^{X_u^p {Y'_c}^T}}{\sum\limits_{c' \in C} e^{X_u^p {Y'_{c'}}^T}} \tag{2.3}$$

2.4.2 Training

First, we describe how training examples are generated from voice assistant usage logs. We create the training examples from the last two weeks of the voice assistant usage logs. Using only the recent portion of the logs ensures that the resolutions suggested by the model are fresh and relevant to new errors, and it will ensure that the amount of training data is small enough that the model can be trained in a reasonable amount of time. We will also filter out the usage logs from the users that issued too little or too many voice commands during that period. Doing so will improve the quality of the training data by removing the parts that are more likely to contain noise.

Given the filtered data, we create one training sample for each pair of user u and voice command c in voice assistant usage logs. Each training sample has a training history H^t which contains every voice command in H_u except c, and a training true label c^t which is the voice command omitted from user history (c). This process has been explained in

Algorithm 1: Create Training Samples

```
Samples = [];
for $u \in U$ do
    for $c \in H_u$ do
        $H^t = H_u - c$;
        $c^t = c$;
        Samples.add($[H^t, c^t]$);
    end
end
```

algorithm 1. Training samples created by this process are randomly shuffled and batched together for the model optimization.

To train our proposed model, we first need to propose a loss function. In this work, we use the cross-entropy loss of $P(\cdot|H^t)$ against c^t to optimize our model. For a training batch B, the loss function can be computed from equation 2.4 which can be used alongside a stochastic optimizer to train the model.

$$\frac{1}{|B|}\sum_{H^t,c^t\in B} -log\left(\frac{P\left(c^t|H^t\right)}{\sum\limits_{c'\in C} P\left(c'|H^t\right)}\right) = \frac{1}{|B|}\sum_{H^t,c^t\in B} -X_{u^t}{Y'_{c^t}}^T + log\left(\sum_{c'\in C} e^{X_{u^t}{Y'_{c'}}^T}\right) \quad (2.4)$$

2.4.3 Collaborative Error Resolution

As we discussed earlier, the item-item similarity measure extracted from this model is the basis of our proposed error resolution method. To compute this measure, we use the latent space representations (matrix Y to be exact) of the voice commands. Since the model we used here is based on Word2Vec model [29], two voice commands will have similar representations if they either appear frequently together or they appear interchangeably in a similar context. This characteristic is particularly useful for error resolution because 1) an error and its resolution frequently appear together in the usage logs data when a user manages to successfully avert that error, and 2) an error and its resolution appear frequently in a similar context since the context only depends on the taste and preferences of the user.

To compute the item-item similarity [23], we use embedding vectors from matrix $Y_{|C|\times d}$. Given the latent space projections extracted from matrix Y, we define the similarity of two voice commands as the cosine of their corresponding vectors:

$$sim(c_1, c_2) = \cos(Y_{c_1}, Y_{c_2}) = \frac{Y_{c_1} \cdot Y_{c_2}}{|Y_{c_1}| \cdot |Y_{c_2}|} \tag{2.5}$$

For a given error e, we define its resolution as the voice command c with the highest similarity score to e that is not an error itself. Note that we can identify whether a voice command is an error or not based on methods described in section 2.3 and we do not need any annotated data. Furthermore, we define the resolution probability of error e as the cosine similarity between its latent vector representation and the vector representation of its resolution.

2.5 Results: real-world errors resolved

In this section, we report the results of running our model on the real-world voice command logs gathered from two weeks in 2019. First, a set of potential voice assistant errors were identified based on methods described in Section 2.3. Then, an error resolution model was trained as described in Section 2.4. Afterward, we tried to derive valid resolutions

Row	Error	Resolution	Resolution prob.
1	Pham	Fam	0.964
2	Mickey Mouse Club	Mickey Mouse Clubhouse	0.96
3	Oh	Old	0.92
4	Cing	Scene	0.938
5	Boat	Bolt	0.948

Table 2.1: Most frequent errors along with their resolution and the error resolution probability.

for the identified errors (following Section 2.4.3). Although we have introduced an approach to automatically detect and resolve voice assistant errors, we still need to rely on human annotators to verify the output of our model and prevent new errors from being introduced to the system. However, the number of errors identified by our proposed approach can sometimes be more than what human annotators can review in a reasonable amount of time. Therefore, we need a mechanism to prioritize the errors for human annotators. In this section, we described the different methods of prioritizing the errors and some examples of real-world, high-impact errors that were prioritized by each method. All the errors reported in this section have been automatically detected and resolved from the voice assistant production environment. As can be seen in the results in this section, most of the errors reported by our model have been identified and resolved correctly. Therefore, the utilization of our proposed model has allowed QA annotators to fix more errors in a shorter amount of time.

Common Errors: Most frequent errors are the ones users notice the most and therefore impact the overall system more than others. such errors with the highest frequency are reported in Table 2.1. The most common error that voice remote users came across

Row	Error	Impacted commands
1	BC	NBC, ABC
2	Mickey Mouse Club	Mickey Mouse Clubhouse, Mickey Mouse
3	Mark	Hallmark
4	CVS	CBS
5	Soprano	Sopranos

Table 2.2: Most impactful errors along with the voice commands they are affecting.

during this evaluation period was the voice command transcribed as “Pham”. In this case, the error resolution model correctly predicted “Fam” to be what the user intended to say (with the confidence of 0.964). At the time of this evaluation, Fam was a newly released TV show and the ASR system was not trained to produce that word and instead incorrectly transcribed the audio. Three out of the five errors in Table 2.1 (“Pham”, “Cing”, and “Boat“) are examples of ASR errors. Once verified by a human annotator, an action item can be taken based on this table. For example, a short-term solution might be to automatically override all occurrences of “Pham” with “Fam” in the NLP system. A longer-term solution might be to retrain the ASR system with sufficient training data corresponding to positive audio examples of the command ”Fam”.

Errors similar to common voice commands: Although frequent errors are experienced by users many times, they are not necessarily the ones that impair the user experience the most. Alternatively, we argue that the errors occurring while using the core capabilities of the system are the most annoying to the user. In other words, users may care a lot more not to experience any errors when they are trying to issue one of the most common commands (such as “tune to MSNBC”) than not experiencing a frequent error. We define a new metric that computes the cosine similarity between vector representations of a given command and any of the top frequent voice commands (not the frequent errors, but the frequent voice commands in general). In this work, we consider an error impactful if it is among the 5 most similar commands (using Equation 2.5) to one of the 100 most popular voice commands. In contrast with error frequency, which is a metric aggregated across all users, this metric measures the impact on individual users. Table 2.2 shows a few examples of such errors. This list highlights the most impactful errors and the voice commands they are affecting. For example, the command "BC" will result in an error, and it happens a lot when the user wants to say either "NBC" or "ABC". Meanwhile, "BC" is an example of an error that is inherently ambiguous. It is most likely due to the user speaking before they activate the microphone on the voice remote (resulting in an incomplete audio signal), so we cannot be sure if the user's true intention was to say "NBC" or "ABC".

While the prioritization methods we have described quantify the impact of errors, many times there is no resolution due to the ambiguity of the command. For example, one of the most common errors in voice command logs is the command "Go to" which is probably caused by users not completing the command. Presenting such errors to human annotators is a waste of time since there is no resolution. To de-prioritize such cases, we also describe some prioritization methods that involve the correctness of errors and their resolutions

Errors likely to be detected correctly: Based on our observations, a voice command's repetition frequency (equation 2.1) is highly correlated with its likelihood of being an error. Therefore, we use equation 2.1 as one of the methods to prioritize the errors for

Row	Error	Repetition prob.	Resolution
1	Khlu	0.51	Clue the Movie
2	Hey Dougie	0.413	Hey Duggee
3	Top wings	0.268	Top wing
4	Boat	0.257	Bolt
5	2 1/2 men	0.235	Two and a Half Men

Table 2.3: Errors most likely to be detected correctly, along with their resolutions.

Row	Error	Resolution	Resolution prob.
1	Any back	Eddie Mack	0.977
2	PS kids	TBS kids	0.976
3	Dsckid	TBS kids	0.976
4	Schedule recordings	Schedule recording	0.974
5	Free horror movie	Free horror movies	0.974

Table 2.4: Errors with the highest resolution probability, along with their resolutions.

human annotators. Table 2.4 lists the five errors with the highest repetition rate (i.e., the likelihood that the command will be repeated in a user session). For example, 51% of the time a user issued a voice command that got transcribed as “Khlu”, it was repeated in the same user session. The error resolution model correctly predicted that the user intended to say “Clue the Movie”. Errors 1, 2, and 4 are due to incorrect ASR — these examples highlight the importance of the error resolution since it is quite difficult for a human annotator to guess what the user intended to say without help from the model. The third error is a plurality mismatch between the user command and the actual title. The last error is due to the NLP system being sensitive to how numbers are spelled out.

Errors likely to be resolved correctly: Errors for which we have found a resolution with a high degree of confidence are essentially the low-hanging fruits. Although these

Row	Cluster of Errors	Resolution
1	FAM Pham, Pham, TV show Pham, Fama	Fam
2	Tell outhouse, The lab House, The wild House, The last house	The Lighthouse
3	BritBox, Board box, Turn box, Redbox	Bird box
4	Civi, CS, CVS, CB	CBS
5	Our America, American, All-Americans, The All-American	All American

Table 2.5: Top clusters of errors and their resolution

errors might not be as frequent or as impactful as others, they are usually the ones that are most likely going to be approved by human annotators and therefore will allow us to better utilize the time spent by annotators. Table 2.4 lists the five errors with highest resolution probability (i.e., error resolution model is most confident). The first three errors are all related to ASR and would require retraining the ASR system with more training data that correctly labels the audio signals corresponding to these errors. Errors 4 and 5 are caused by the NLP system being too rigid with how it matches the command to a menu title.

Errors appearing in clusters: One of the most interesting observations we had during our experiments was that many times multiple errors and one valid voice command would appear very close to each other in the latent space learned by the collaborative filtering model. Based on our observations, this happens mostly because of ASR issues. For example, when users try to say “Bird Box” (name of a movie) to the voice assistant, the ASR will miss-transcribe in many different ways such as “BritBox”, “Board box”, “Turn box”, “Redbox”, etc. Such ASR mistakes will cause multiple different formats of the error to appear in the voice command logs and to be projected very close to each other in the latent space. In this work, we define a valid voice command and a list of errors as a cluster of errors, when there are more than 3 errors among the 5 most similar commands to a valid voice command. We also define the frequency of a cluster as the sum of the frequencies of all the errors in that cluster and sort the clusters based on their frequencies. The most frequent cluster will be sent to human annotators. Table 2.5 shows the list of top five error clusters, sorted by the sum of the frequencies of the errors in each cluster. This table highlights how our framework automatically identifies the same spoken command being transcribed in different ways. Clusters 1, 2, 3, and 5 are related to TV shows and movies that have been released relatively recently. The fourth cluster shows various transcriptions where the intention was likely “CBS”. The solution to all of these errors would involve retraining the ASR system with data targeting these words.

2.6 Conclusion and Future Work

In this work, we introduced the voice assistant error resolution framework which is based on a collaborative filtering model. We have explained how voice command logs can be leveraged to tackle voice assistant errors and we have shown the effectiveness of our proposed approach against a large-scale, state-of-the-art voice assistant. Although this work was highly focused on a single voice assistant, all the models, and processes explained here can be applied to any voice assistant.

Currently, this framework is being used as a supportive tool for Quality Assurance engineers to identify the existing errors and come up with action items. This allows a very targeted quality assurance process without the need for annotators going through millions of commands every day. Even better, the output of this framework can be used to directly generate training data for the voice assistant and potentially fix errors without any human intervention. Integrating this framework into the training pipeline of the voice assistant can help reduce the turnaround time even further. We will also explore the possibility of tightly integrating the collaborative error resolution model directly into the NLP platform as a fallback mechanism for handling errors. Whenever the NLP system is incapable of handling a voice command, it can check to see if the error resolution model has a valid resolution for that command. Without this tight integration, the voice assistant will keep failing on that command until training data is created by the error resolution framework and the voice assistant is retrained with that data. However, the tight integration will act as a temporary fix for the error until the voice assistant is retrained.

Chapter 3: Adversarially Robust Cold-Start Recommendations

3.1 Introduction

Recommendation systems are an integral part of our everyday lives. Online stores, video streaming services, and news platforms are just a few examples of the popular services that rely on recommendation systems. In these systems, a high-quality recommendation system increases the number of user engagements, which in turn affects the revenue of the business. Additionally, the recommendation system determines how often an item is shown to the users, which in turn affects the content providers' and sellers' revenue. Therefore, recommendation systems are considered business-critical to many companies. Since recommendation systems directly influence billions of dollars in revenue, malicious actors have a big incentive to attack those systems. For example, a business rival might try to decrease the revenue of the platform owner by causing poor recommendations. Another example is a seller or a content provider that might try to increase their revenue by manipulating the system to recommend their product more frequently and to more users.

Adversarial attacks refer to a family of attacks against machine models in which a malicious actor makes small changes to the input of the model to cause large changes to the output. For example, in the computer vision domain, an adversary can slightly change an image to cause it to be misclassified by the machine learning model while keeping the change unnoticeable to human eyes. There have been multiple studies on the susceptibility of machine learning models to such attacks in both academic and industrial settings. In particular, some studies have shown that the machine learning models used in real-world systems such as self-driving cars [1], industrial copyright detection systems [30], and high-frequency trading systems [12] are susceptible to adversarial attacks.

Given the wide adoption of machine learning models in recommendation systems in the recent years [27, 31, 32] and the susceptibility of machine learning models to adversarial attacks, there is a clear need for studying such attacks against recommendation systems. Christakopoulou and Banerjee [33] studied the susceptibility of recommendation models to poisoning attacks where adversarially generated user profiles are injected into the recommendation system's training data to manipulate its behavior. To evaluate how realistic a poisoning attack is, one can measure the number of malicious profiles that must be injected into training data before the attack succeeds. For example, the attacks described by Christakopoulou and Banerjee [33] require as many as 6.3% of the user profiles to be generated by the adversary. For a real-world large-scale system with hundreds of millions of users, that would require creating millions of fake user profiles by the adversary. Furthermore, in many cases creating a fake user profile for a real-world system would require paying a subscription fee per user — for example, both Netflix and Amazon Prime Video required a paid subscription — or it would require purchasing items to create the user-item interactions for a fake user profile. The financial cost of such attacks would be significantly greater than the incentives of a typical adversary, and so this threat model may not be realistic for large scale systems.

To address the limitations of user profile poisoning attacks, in this work we propose a new attack model that targets cold-start recommendations. In particular, we look at content-based methods [34, 35, 36] where the content information — such as description, images, tags, etc. — of a new item is used for generating recommendations. In our proposed attack, a malicious actor can achieve their desired outcome by making small perturbations to the content information of a new item while making sure the perturbations are not noticeable by a human. Specifically, we propose an attack model in which a new item's content information will be perturbed by a gradient-based method before it is added to the recommendation system.

In comparison to fake user profile poisoning attacks, this attack model has a few advantages for the attacker. First, content data such as text or images can be attacked by gradient-based approaches while the fake user profile attacks rely on zeroth-order optimization. Generally speaking, gradient-based optimization problems are easier to solve, which in turn makes the gradient-based attacks more effective compared to zeroth-order methods. Second, it is relatively easy for a content provider or a seller to perform this attack on a real-world system. Many of the platforms allow their content providers or sellers to upload their items' content information directly into the system which makes them susceptible to such attacks. Finally, our proposed attack model does not require any fake profiles or fake purchases, and hence there are no excessive financial costs associated with them. The effectiveness, simplicity, and the low-cost nature of such attacks make them particularly harmful to real-world recommendation systems.

Given the significance of recommendation systems to many businesses, it is essential to train machine learning models that are robust to such attacks. One of the most common defense approaches against adversarial attacks is adversarial training [9]. In this approach, instead of training the model on the raw training data, the model is trained on the adversarially perturbed version of the training set. In other words, at each training iteration, the input training data is adversarially perturbed on-the-fly and the model is trained on perturbed inputs along with the true labels. Given the success of this approach in training adversarially robust models in other domains, we propose a defense mechanism against our attack based on this approach.

The main contributions of this work are: **(1)** We study a new attack model against cold-start recommendation systems that can successfully be used in real-world settings. **(2)** We also discuss the financial and legal ramifications of such attacks in real-world systems, and how a malicious actor can achieve their desired outcomes in real-world systems using this approach. **(3)** We evaluate this attack model against a large-scale real-world recommendation dataset and show the effectiveness of such attacks in a real-world setting. **(4)** We propose how adversarial training can be used to create models that are more robust against such attacks and evaluate the effectiveness of this defense approach on a real-world dataset.

3.2 Cold-Start Model

The cold-start problem refers to the situation where we don't have access to the historical user-item interaction data for a newly added item or a newly joined user. Such situations are common in many real-world systems where new items and new users are frequently added to the system. For example, new items are added daily to online stores, and new video content is continuously uploaded to video streaming platforms. Since many recommendation models rely on user-item interaction data, we need other approaches for handling these newly added items or users.

One of the most common approaches for tackling the cold-start problem is using content-based methods [34, 35, 36]. In content-based methods, the *content information* of a new user — such as age, location, etc. — or a new item — such as description, image, etc. — is used to generate recommendations. Since the content information does not depend on the user-items interactions and are usually available from the moment an item or a user is added to the system, content-based recommendation models are a great fit for the cold-start problem. In recent years, many machine learning based approaches [37, 38, 39, 40, 41] have emerged to tackle the cold-start problem by leveraging both the content data as well as user-item interaction data. These models work by learning a function that projects the content information of an item — or a user — into the latent space of a collaborative filtering model. These *derived* embeddings are then used to generate recommendations for new users and new items.

In this work, we use a similar approach: we train a collaborative filtering model on historical user-item interaction data and we train a neural network that projects the content information of cold items into the collaborative filtering model's embedding space.

3.2.1 Collaborative Filtering model

First, we train a collaborative filtering model on the historical user-item interaction data. This model will be responsible for predicting future user-item interactions based

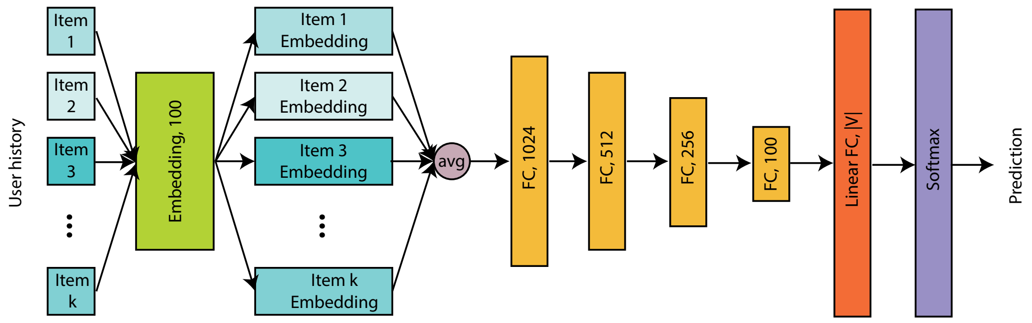


Figure 3.1: The Collaborative Filtering model trained on historical user-item interaction data

on the history of previous user-item interactions. In this work, we have adopted an implicit-feedback model inspired by the models proposed by Barkan and Koenigstein [26], Covington et al. [27]. Figure 3.1 shows the high-level architecture of this model.

This model tries to predict the probability of a user interacting with an item based on the previous items that the user has interacted — watched, purchased, etc. — with. To do so, the first step is to retrieve the history of the items a given user u has previously interacted with. We call this history $H(u) = \{item_1, item_2, \cdots, item_k\}$ where k is the number of items the user has interacted with. Next, we pass the history $H(u)$ though an embedding lookup layer to get the embeddings of all the items in the user history. For a given item i, we refer to that item's embedding as $v_i \in \mathbb{R}^d$. Note that the embedding layer is not pre-trained and is randomly initialized and trained with the rest of the model. Then, we compute the average of the embeddings of items in $H(u)$ to get a single vector representation for the user history (equation 3.1).

$$\frac{1}{|H(u)|} \sum_{i \in H(u)} v_i \tag{3.1}$$

This vector is then passed through multiple layers of fully connected feed-forward layers with $tanh$ activation function. Next, the output of the last non-linear layer is passed through a fully connected linear layer with $|V|$ neurons where V refers to the set of all items in the system — i.e. vocabulary of the items. This layer predicts one score per item in vocabulary that corresponds to the likelihood of the user u interacting with that item. The output of this layer is then passed into a Softmax layer to get the probability distribution of the user u interacting with each of the items in vocabulary. This probability distribution can then be used for different recommendation tasks. For example, the items with the highest probability can be used in Top-N recommendations.

Model training. To train this model, we use the held-out watch approach proposed by Covington et al. [27]. In this approach, for each user with history $H(u) = \{item_1, item_2, \cdots, item_k\}$ we create k different training examples: one example per item in the history. In each example, we remove one of the items from the history and use it as the prediction label, and use the rest of the $k-1$ items as the inputs to the model. We then shuffle these training examples and do batch training. Using the held-out item as the true classification label, we train the model by optimizing a cross-entropy loss function over the output of the Softmax layer. Note that in this work we do not use negative sampling.

Item-Item similarity. One of the most commonly used recommendation tasks in real-world systems is recommending things like ”customers who bought this also bought” or ”because you watched”. In these scenarios, a seed item — such as a product the user is looking at or a movie the user has just watched — is used to generate recommendations for other items that are similar to this seed item. One of the most common approaches to power this experience is recommending the items with the highest cosine similarity score to the seed item [42]. In our model, we can take a similar approach and define the similarity of two items as the cosine of their corresponding embedding vectors (equation 3.2).

$$similarity(i, j) = cos(v_i, v_j) = \frac{\overrightarrow{v_i} \cdot \overrightarrow{v_j}}{||\overrightarrow{v_i}|| \times ||\overrightarrow{v_j}||} \tag{3.2}$$

Adding a new item to the system. In this model, the only parts that depend on the item vocabulary are the initial embedding layer — i.e. the green layer in figure 3.1 — and the final fully connected linear layer — i.e. the dark orange layer in figure 3.1. Since the last linear layer is essentially a $d \times |V|$ matrix of d-dimensional embeddings for the items in vocabulary, we can consider both layers as embedding layers. Therefore, to add a new item to the system we just need to append one more embedding vector to each of these two layers and initialize that vector with a good representation of the new item.

3.2.2 Content based model

To add new items to the system, we must be able to initialize the new item's embedding. To do so, we will use another model that projects the content information of a given item into the collaborative filtering model's embedding space. There are many different modalities for the content data of an item such as title, image, description, tags, price, reviews, etc. Here, we focus on only one modality of the content data: image.

In this work, we use a convolutional encoder for the content-based model. This model will be responsible for learning a mapping from image space to the collaborative filtering model's embedding space. We train the model using the images associated with the items in the vocabulary and their pre-trained embeddings. Figure 3.2 shows the

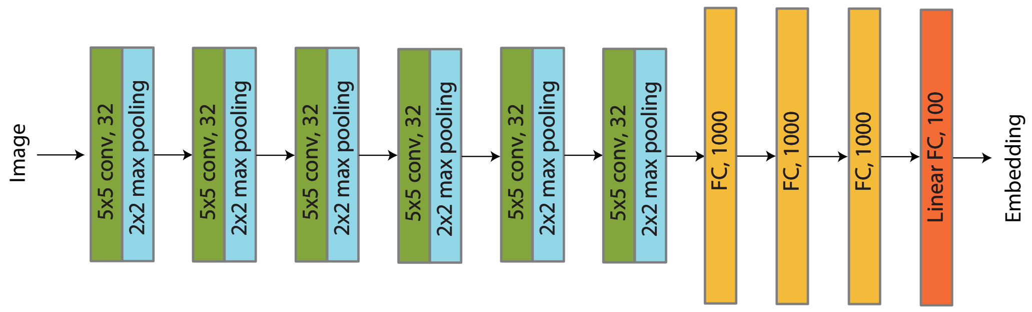


Figure 3.2: Convolutional encoder to project movie posters into their embedding.

architecture of this model.

First, the image is passed through multiple pairs of a convolution layer followed by a max-pooling layer. Then, the output of the last max-pooling layers is flattened and passed through a few fully connected feed-forward layers with *relu* activation functions. Finally, the output of the last fully connected layer is passed through a linear layer with the same number of neurons as the collaborative filtering model's embedding dimension. The output of this layer can be treated as the projection of the item's image into the collaborative filtering model's embedding space. For an item i, we refer to the projected embedding derived from item i's image as $f(x_i;\theta)$ where x_i is the item i's image, $f(\cdot)$ is the function described by our proposed convolutional encoder model, and θ is the parameters of the model.

We train this model by maximizing the cosine between an item's pre-trained embedding and the projected embedding derived from its image (equation 3.3).

$$L(V;\theta) = \frac{-1}{|V|} \sum_{x,v \in V} \frac{f(x;\theta) \cdot v}{\|f(x;\theta)\| \times \|v\|} \tag{3.3}$$

Whenever a new item is introduced to the system, we can compute that item's projected embedding using this model and set that value for its embedding in the collaborative filtering model.

3.3 Adversarial Attacks on Cold-Start Recommendation Models

Adversarial attacks against machine learning models have been extensively studied over the past few years. These attacks try to manipulate the output of a machine learning model by making very small but deliberately chosen perturbations to the input data. Because of the small perturbations used in these attacks, the attack usually goes unnoticed which poses a great threat to many machine learning models. Previous works have shown that adversarial attacks can be used to successfully attack many sensitive systems such as self driving cars [1], industrial copyright detection systems [30], facial recognition systems [43], object detectors [2, 3, 4], speech recognition systems [5], and high frequency trading systems [12].

There are two major types of adversarial attacks: targeted and non-targeted. In a non-targeted attack, the adversary only tries to make sure that the output of the perturbed input is as different as possible from the true output. The optimization problem in equation 3.4 describes the non-targeted adversarial attacks. In this equation, δ refers to the perturbation, x is the input to the model, y is the true output, θ is the set of parameters that define the model, L is the loss function, and ϵ defines a bound for how big the perturbations can be.

$$\max_{\delta} L\left(x+\delta, y ; \theta\right) \qquad s.t. \left\|\delta\right\|_{p} \leq \epsilon \tag{3.4}$$

On the other hand, in targeted attacks, the adversary tries to make the output of the model as similar as possible to a particular target. Similar to non-targeted attacks, we can formulate targeted adversarial attacks with an optimization problem as described in equation 3.5. Here, y_t refers to the target output for the attack.

$$\min_{\delta} L\left(x+\delta, y_t; \theta\right) \qquad s.t. \left\|\delta\right\|_p \leq \epsilon \tag{3.5}$$

In this work, we assume the adversary controls the image associated with an item and can adversarially perturb it before uploading it into the system. This is a realistic assumption given that in many real-world systems the seller or the content provider directly uploads the content information of their item. Using a targeted adversarial attack, an adversary can essentially manipulate their item's embedding however they want while keeping the perturbations small enough that they would go unnoticed. For example, the adversary can make sure that their item's embedding is almost identical to the embedding of a popular item which would result in their item getting recommended to a majority of the users.

To this end, we propose a targeted adversarial attack against our content-based model. Equation 3.6 describes the optimization problem that needs to be solved for this attack. Using this optimization, the adversary can find a small perturbation δ to their item's image x such that the output would be very similar to the target embedding v_t.

$$\max_{\delta} \frac{f\left(x+\delta; \theta\right) \cdot v_t}{\left\|f\left(x+\delta; \theta\right)\right\| \times \left\|v_t\right\|} \qquad s.t. \left\|\delta\right\|_\infty \leq \epsilon \tag{3.6}$$

There are many different ways to solve this optimization problem. In this work, we use projected gradient descent (PGD) [9] to solve this optimization problem. In this approach, we iteratively update δ via a gradient descent step and then project it back into the ℓ_∞ ball with radius ϵ. Algorithm 2 describes this process. In this process, we first compute the gradient of function l (cosine of the angle between $f(x+\delta;\theta)$ and v_t) with respect to δ. We then take a gradient step in the direction of sign(g) with the step size of ϵ_s. Finally, we clip δ back into the ℓ_∞ ball with radius ϵ. We repeat this process for K iterations to create stronger attacks.

Algorithm 2: Creating targeted adversarial example

```
$\delta \leftarrow 0$;
for $k \leftarrow 1$ to $K$ do
    $g \leftarrow \nabla_\delta l(x+\delta, v_t; \theta)$;
    $\delta \leftarrow \delta + \epsilon_s \cdot \text{sign}(g)$;
    $\delta \leftarrow \text{clip}(\delta, -\epsilon, \epsilon)$;
end
```

3.3.1 Real-world attacks

Using our proposed adversarial attack model, there are many different ways that an adversary can create realistic attacks. In this part, we propose a few of such attacks, discuss the adversary's incentives for such attacks in real-world settings, and describe how our proposed optimization problem can be used for such attacks.

Attacking popular items. In many platforms, the revenue gained by a seller or a content provider is directly impacted by their item's sales or user engagement numbers. Also, it is well-known that the higher an item appears on the list returned by a recommendation system, the higher the chances are that users will interact with that item. Therefore, there is a strong financial incentive for a malicious actor to try to trick the recommendation system into ranking their items higher. Given that the popular items are the ones that are recommended most frequently to the users, the malicious actor can gain a high recommendation ranking by making their item very similar to a popular item. Using our proposed targeted attack in equation 3.6, the adversary can target the embedding of a very popular item (v_t) in their attack. By doing so, their item's embedding is going to be very similar to the popular item's embedding which in turn results in the new item getting recommended anywhere the popular item is recommended.

Attacking sensitive items. Many times, there are sensitive or protected items on a platform. For example, kids' contents are always under a lot of scrutiny on video streaming platforms. To keep a high-quality experience for the kids — and many times due to legal requirements — it is of vital importance for the streaming services to make sure that content inappropriate for kids is not recommended to them. Furthermore, many times there are contractual obligations imposed by the content provider on the streaming service. For example, some kids' content providers impose limitations on what other content can appear next to their content. Therefore, the streaming platform can be in breach of contract and suffer large financial damages if a content that is inappropriate for kids appears next to a kids' content. Consequentially, a malicious actor such as a business rival can cause great financial damages by attacking one of these protected items. For example, the malicious actor can upload a horror movie into the system and create an adversarial example from that horror movie targeting one of the kids' contents. By doing so, the horror movie would effectively get recommended anywhere the kids' content was recommended.

Targeting groups of users. An adversary can also target a group of users. The incentive behind such an attack can be either causing harm to the platform — e.g., by targeting the children with horror content — or gaining financial benefit — e.g., by targeting a specific section of the population with a product. To do so, the adversary can find a set of items that are popular among that particular group of users and then try to attack all those items simultaneously. Equation 3.7 describes such an attack where V_t is the set of items that best represent the preferences of the targeted group.

$$\max_{\delta} \frac{1}{|V_t|} \sum_{v' \in V_t} \frac{f(x+\delta;\theta) \cdot v'}{\|f(x+\delta;\theta)\| \times \|v'\|} \qquad s.t. \; \|\delta\|_{\infty} \leq \epsilon \tag{3.7}$$

3.4 Adversarilly Robust Model

A machine learning model is considered adversarially robust if it can maintain a reasonable amount of its accuracy against adversarial attacks. The problem of training models that are robust to adversarial attacks has been studied for different types of adversarial attacks [9, 10]. Generally speaking, these approaches train the model by not just trying to minimize the loss function, but rather trying to minimize the loss function in the presence of deliberately chosen adversarial perturbations.

Given the critical nature of recommendation systems, it is important to explore how we can train models that are robust to such attacks. Adversarial training [9] is one of the techniques for training a model that is robust to adversarial attacks. Adversarial training can be viewed as a saddle point optimization problem where we are trying to find a set of parameters that can minimize the loss value on the training data no matter what perturbations are added by the adversary. Equation 3.8 describes the general form of such a min-max optimization where $\mathcal{D}$ is the training dataset and L is the loss function of the model. The solution to this saddle point problem is a robust model.

$$\min_{\theta} \mathbb{E}_{(x,y)\sim\mathcal{D}} \left[\max_{\delta} L\left(x+\delta, y; \theta\right)\right] \qquad s.t. \left\|\delta\right\|_p \leq \epsilon \tag{3.8}$$

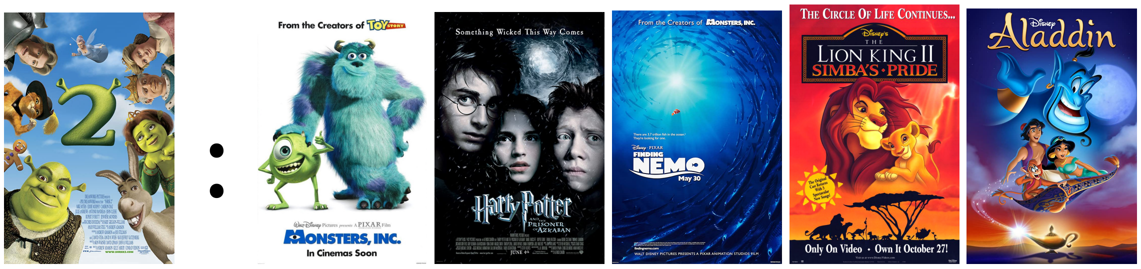
Using this framework, we can define the saddle point optimization problem for our content based model as equation 3.9.

$$\max_{\theta} \frac{1}{|V|} \sum_{x,v\in V} \min_{\delta} \frac{f\left(x+\delta;\theta\right)\cdot v}{\left\|f\left(x+\delta;\theta\right)\right\| \times \left\|v\right\|} \qquad s.t. \left\|\delta\right\|_\infty \leq \epsilon \tag{3.9}$$

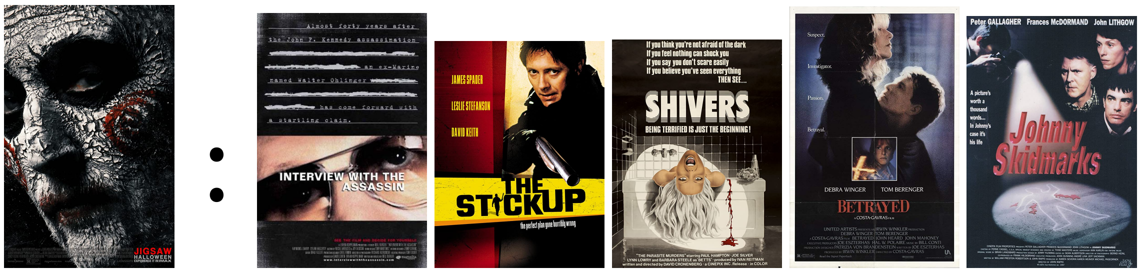
Algorithm 3: Adversarial Training

```
for $epoch \leftarrow 1$ to $N$ do
    for $batch\ B \subset V$ do
        $B_{\text{adv}} \leftarrow \{\}$;
        for $x, v \in B$ do
            $\delta \leftarrow 0$;
            for $k \leftarrow 1$ to $K$ do
                $g_{\text{adv}} \leftarrow \nabla_\delta l(x+\delta, v; \theta)$;
                $\delta \leftarrow \delta + \epsilon_s \cdot \text{sign}(g_{\text{adv}})$;
                $\delta \leftarrow \text{clip}(\delta, -\epsilon, \epsilon)$;
            end
            $B_{\text{adv}} \leftarrow B_{\text{adv}} \cup \{(x+\delta, v)\}$;
        end
        $g_\theta \leftarrow \mathbb{E}_{(x,y)\in B_{\text{adv}}} [\nabla_\theta l(x, v; \theta)]$;
        $\theta \leftarrow \theta - \tau g_\theta$;
    end
end
```

As discussed by Madry et al. [9], one way for solving this optimization problem is using stochastic gradient descent and computing the ∇_θ at the maximizer of the inner


2
From the Creators of Toy Story
Monsters, Inc.
In Cinemas Soon
Something Wicked This Way Comes
Harry Potter and the Prisoner of Azkaban
From the Creators of Monsters, Inc.
Finding Nemo
May 30
The Circle of Life Continues...
The Lion King II Simba's Pride
Only On Video • Own It October 27!
Aladdin

(a) Most similar items to the move Shrek 2 — an in-vocabulary item.



(b) Most similar items to the move Jigsaw — an out of vocabulary (cold) item.

Figure 3.3: Most similar items to two examples of in-vocabulary and out of vocabulary items.[1]

function. That means at each iteration of the gradient descent we have to find the maximizer of the inner function, compute the gradient of the loss function with respect to θ, and take a step in that direction. To find the maximizer of the inner function, we can use the same approach as we discussed for generating adversarial examples — using an iterative PGD method. Algorithm 3 describes the K-step PGD adversarial training that we use in this work. Note that the ϵ used for training a robust model can be different than the ϵ used in creating adversarial examples.

The model resulted from the adversarial training approach is significantly more robust against adversarial attacks compared to a naturally trained model — i.e. trained on minimizing the loss function in absence of any adversarial perturbations.

[1]Figure 3.3: Shrek 2 ©2004 Dreamworks LLC. All Rights Reserved. Monsters, Inc. ©2001 Disney/Pixar. All Rights Reserved. Harry Potter and the Prisoner of Azkaban ©2004 Warner Bros. Ent. Harry Potter Publishing Rights J.K.R. Finding Nemo ©2003 - Pixar/Disney. All Rights Reserved. The Lion King 2: Simba's Pride ©1998 - Walt Disney Studios. All rights reserved. Aladdin ©1992 Walt Disney Productions. All Rights Reserved. Jigsaw ©2017 Lions Gate Entertainment Inc. All Rights Reserved. The Stickup ©2002 Promark Entertainment Group. Shivers ©1975 Cinepix. Betrayed ©1988 Metro-

3.5 Experiments

To study the effectiveness of our proposed attack, we evaluate our approach against a large scale real-world dataset.

3.5.1 Dataset

In this work, we evaluate our approach against the Netflix dataset [44]. This dataset contains more than 100 million ratings for over 17 thousand movies by 480 thousand users. The large size of this dataset makes it a perfect candidate for evaluating our approach in real-world settings. Furthermore, since in real-world systems implicit feedback from the users is significantly more frequent than explicit feedback, in this work we will also train our model on implicit feedback data. Following the same approach as in previous works [45, 46], we extract positive ratings (rating=4 or 5) from the dataset.

We also use the posters for the movies in the Netflix dataset to create training data for our content-based model. Given that most neural network architectures that process image data only work with a fixed-sized input, we crop and down-sample the posters to 300×300 pixels.

3.5.2 Naturally trained model

Using the Netflix dataset, we first train the collaborative filtering model. While this model has been trained to predict a user's behavior and preferences based on their history, it learns high-quality embeddings for all the movies in the training vocabulary. Using the

Goldwyn-Mayer Studios Inc. All Rights Reserved. Johnny Skidmarks ©1998 Cinepix.

method described in section 3.2.1, we can drive item-item similarity scores using these embeddings. For example, figure 3.3a shows the most similar movies to the movie Shrek 2. All of the 5 most similar movies to Shrek 2 are also kids' movies, and 4 of them are animations as well.

Then, we train the content-based model using the posters of the movies in the Netflix dataset along with their corresponding embeddings learned by the collaborative filtering model. Using this model we can compute the embedding vector of a new movie using its poster, and then inserting that movie along with its embedding into the collaborative filtering model. For example, using this approach we can add the movie Jigsaw into our system even though it does not exist in the Netflix dataset. Figure 3.3b shows the most similar movies to Jigsaw based on this approach. Although the collaborative filtering model was never trained on the Jigsaw movie, the content-based model has successfully computed an embedding vector for this movie. All of the most similar movies to Jigsaw share the same genres (crime, horror) with this movie.

3.5.3 Attacking the naturally trained model

We evaluate the robustness of the naturally trained model against the adversarial attacks proposed in section 3.3 by running a set of experiments with targeted attacks. In each experiment, we randomly choose a movie and remove that movie form the collaborative filtering model to create a cold-start situation. We then randomly choose a second movie from the collaborative filtering model's vocabulary as the attack target. Using algorithm 2, we create an adversarial example from the cold-start item against the

	ϵ=2	ϵ=4	ϵ=6	ϵ=8
N=1	25.59	74.02	95.51	99.02
N=2	28.13	77.54	95.9	99.22
N=3	29.3	79.1	96.48	99.22
N=4	30.86	80.66	96.88	99.41
N=5	31.84	81.64	97.27	99.41

Table 3.1: Success rate of adversarial attacks against naturally trained model

chosen target. Finally, we evaluate the success of this attack by finding the top-N most similar items to the targeted item and evaluating whether or not the adversarial example is among them.

To evaluate the attack, we do a grid-search on the hyperparameters of the proposed attacks. We run experiments with $\epsilon \in \{2.0, 4.0, 6.0, 8.0\}$ and attack iterations $K \in \{20, 40, 60, 80, 100, 1000\}$. We also evaluate the success rate of our attacks — i.e. the percentage of the times that the adversarial example is among the top-N most similar items to the target — for different values on $N \in \{1, 2, 3, 4, 5\}$. Furthermore, we repeat each experiment for 512 randomly chosen pairs of cold-start items and targets.

Based on the evaluation results, it seems like even an attack with 20 steps is essentially as effective as a 1000 step attack. In essence, this means that there is no need for computationally expensive attacks to trick the model. Therefore, in this work, we will only report the highest success rate among attacks with different step counts.

Table 3.1 shows the success rate of adversarial attacks with different ϵ values and at different top-N evaluations against the naturally trained model. Based on these results, an attack with perturbations as small as $\epsilon = 6$ — which is smaller than most other attacks in the computer vision domain — can render the model completely useless. Over 99%

	$\epsilon = 2$				
	@1	@2	@3	@4	@5
Natural	25.59	28.13	29.3	30.86	31.84
Adversarial / $\epsilon = 1$	3.32	3.91	4.69	5.27	5.47
Adversarial / $\epsilon = 4$	1.95	2.15	2.54	**3.13**	**3.32**
Adversarial / $\epsilon = 8$	**1.56**	**2.15**	**2.54**	3.32	3.91

	$\epsilon = 4$				
	@1	@2	@3	@4	@5
Natural	74.02	77.54	79.1	80.66	81.64
Adversarial / $\epsilon = 1$	8.40	8.59	9.18	11.33	11.91
Adversarial / $\epsilon = 4$	5.08	6.25	7.03	8.20	9.18
Adversarial / $\epsilon = 8$	**4.88**	**5.86**	**6.05**	**6.84**	**8.01**

	$\epsilon = 6$				
	@1	@2	@3	@4	@5
Natural	95.51	95.90	96.48	96.88	97.27
Adversarial / $\epsilon = 1$	15.82	18.95	21.48	23.05	23.63
Adversarial / $\epsilon = 4$	**9.96**	**12.11**	**13.87**	**14.26**	**15.23**
Adversarial / $\epsilon = 8$	11.33	12.89	14.45	15.23	16.02

	$\epsilon = 8$				
	@1	@2	@3	@4	@5
Natural	99.02	99.22	99.22	99.41	99.41
Adversarial / $\epsilon = 1$	30.47	33.2	35.35	37.7	38.67
Adversarial / $\epsilon = 4$	**18.75**	**20.31**	**22.07**	**23.83**	**25.00**
Adversarial / $\epsilon = 8$	19.92	22.07	23.63	25.39	27.34

Table 3.2: Robustness of naturally trained model in comparison to adversarially trained models.

(a) Clean image

(b) adversarial perturbations

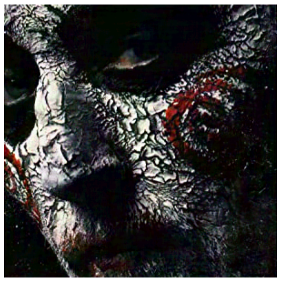

(c) Adversarial example

Figure 3.4: Clean vs. adversrially perturbed posters of a cold item. The adversarial perturbations are not noticeable to humans.

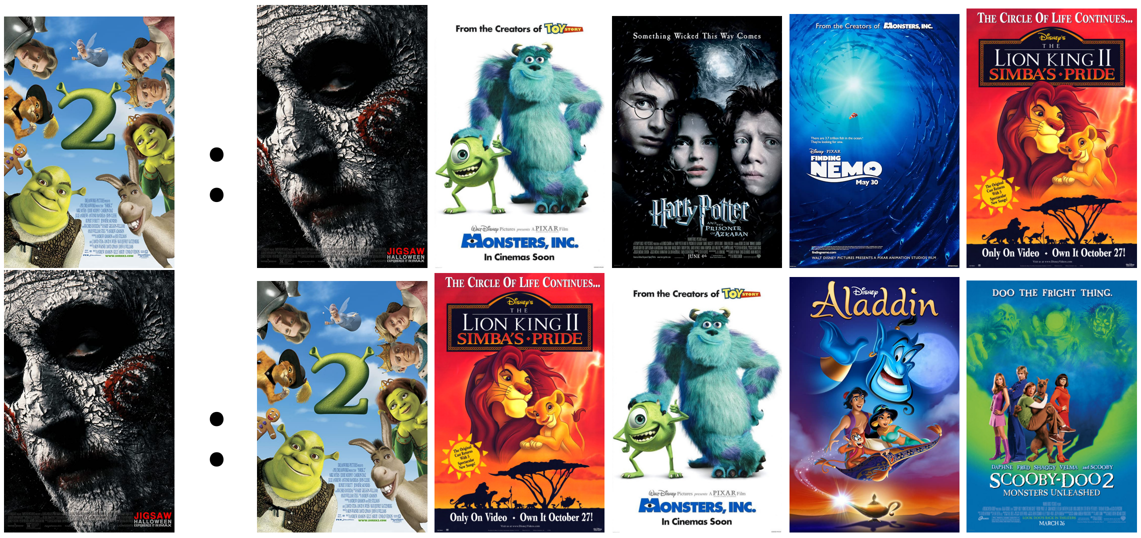


Figure 3.5: Item-item similarity results after an adversarial attack.[2]

of the times an adversary can create a successful attack from any item against any other item.

Furthermore, this attack does not cause any meaningful changes in the images that would impact the users' perception of that item or get noticed by editors, annotators, or quality assurance specialist. Figure 3.4 shows a clean image and an adversarially perturbed image side by side. Humans can't notice any difference between these two images.

[2]Figure 3.5: Shrek 2 ©2004 Dreamworks LLC. All Rights Reserved. Jigsaw ©2017 Lions Gate Entertainment Inc. All Rights Reserved. Monsters, Inc. ©2001 Disney/Pixar. All Rights Reserved.

As an example, we create an adversarial example from the movie Jigsaw by targeting the movie Shrek 2. Figure 3.5 shows the most similar movies to these two movies after the adversarial attack. By using this attack, the adversary has successfully made the movie Jigsaw (a horror movie) the most similar movie to Shrek 2 (a kids' movie). The adversarial example itself has also essentially become a kid's movie since all of the most similar movies to it are other kids' movies.

3.5.4 Adversarially trained models are robust

To evaluate the robustness of adversarially trained models against our proposed attacks, we train 3 new models using the approach described in algorithm 3. We train models with perturbation magnitudes of $\epsilon = 1$, $\epsilon = 4$, and $\epsilon = 8$. For each model, we then run the same set of experiments as the ones for the naturally trained model. We randomly choose two programs as cold-start item and target, we remove the cold-start item from the collaborative filtering model, and then we create an adversarial example from the cold-start item against the target item. We evaluate each model against adversarial attacks with $\epsilon \in \{2.0, 4.0, 6.0, 8.0\}$ and $K \in \{20, 40, 60, 80, 100\}$. We also evaluate each attack's success rate for different values of $N \in \{1, 2, 3, 4, 5\}$ and repeat the experiment for 512 randomly chosen pairs of cold-start items and targets.

Figure 3.6 shows the success rate of attacks with different ϵ at $N = 1$ and $N = 5$ for all the 4 models: naturally trained model and adversarially trained models with $\epsilon = 1$, $\epsilon = 4$, and $\epsilon = 8$. All adversarially trained models are significantly more robust to the

Harry Potter and the Prisoner of Azkaban ©2004 Warner Bros. Ent. Harry Potter Publishing Rights J.K.R. Finding Nemo ©2003 - Pixar/Disney. All Rights Reserved. The Lion King 2: Simba's Pride ©1998 - Walt Disney Studios. All rights reserved. Aladdin ©1992 Walt Disney Productions. All Rights Reserved. Scooby-Doo 2: Monsters Unleashed ©2004 Warner Brothers. All rights reserved.

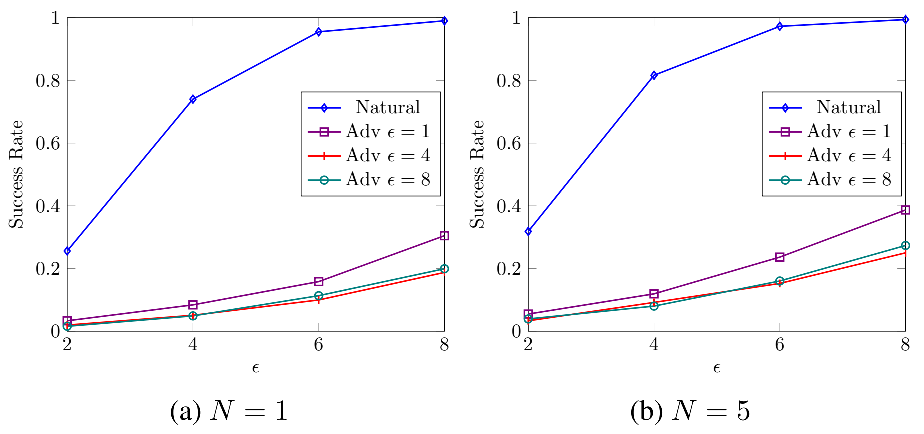


Figure 3.6: Adversarial attack's success rate for N=1 and N=5 at different attack ϵ values.

adversarial attacks compared to the naturally trained model. It also appears that using larger perturbations magnitudes during the adversarial training process can improve the model robustness up to a point.

Table 3.2 contains the success rate of adversarial attacks with different ϵ values against all 4 models. While the naturally trained model is effectively rendered useless against adversarial attacks, the adversarially trained models can maintain a high degree of accuracy. For example, attacks with $\epsilon = 8$ were able to successfully target the top-1 most similar item in 99% of the times for naturally trained model while they only achieved an 18.75% success rate for the best adversarially trained model. In some cases, the successful adversarial attacks can be 9.5 times more in the naturally trained model in comparison to the adversarially trained models.

3.6 Conclusion

In this work, we showed the vulnerability of cold-start recommendation systems to adversarial attacks. Using the attack model proposed in this work, a malicious actor can successfully create adversarial examples in 99% of the times. We discussed how such attacks can happen in real-world systems and also discussed the legal and financial ramifications of such attacks. We also showed how such attacks cannot be noticed by any human and can go undetected in real-world systems. To defend against these attacks, we proposed to train the model by using adversarial training. We showed that using this approach during model training can immensely improve the robustness of the model against such attacks. Evaluating our approach against a real-world dataset shows that adversarially trained models can be up to 9 times more robust compared to naturally trained models.

Chapter 4: Adversarial Attacks on Copyright Detection Systems

4.1 Introduction

Machine learning systems are easily manipulated by adversarial attacks, in which small perturbations to input data cause large changes to the output of a model. Such attacks have been demonstrated on a number of potentially sensitive systems, largely in an idealized academic context, and occasionally in the real-world [1, 2, 3, 4, 5, 6].

Copyright detection systems are among the most widely used machine learning systems in the industry, and the security of these systems is of foundational importance to some of the largest companies in the world. Despite their importance, copyright systems have gone largely unstudied by the ML security community. Common approaches to copyright detection extract features, called fingerprints, from sampled video or audio, and then match these features with a library of known fingerprints. Examples include YouTube's *Content ID*, which flags copyrighted material on YouTube and enables copyright owners to monetize and control their content. At the time of writing this chapter, more than 100 million dollars have been spent on Content ID, which has resulted in more than 3 billion dollars in revenue for copyright holders [47]. Closely related tools such as Google Jigsaw detect and remove videos that promote terrorism or jeopardized national security. There is also a regulatory push for the use of copyright detection systems; the recent EU Copyright Directive requires any service that allows users to post text, sound, or video to implement a copyright filter.

A wide range of copyright detection systems exist, most of which are proprietary. It is not possible to demonstrate attacks against all systems, and this is not our goal. Rather, the purpose of this work is to discuss why copyright detectors are especially vulnerable to adversarial attacks and establish how existing attacks in the literature can potentially exploit audio and video copyright systems.

As proof of concept, we demonstrate an attack against real-world copyright detection systems for music. To do this, we reinterpret a simple version of the well-known "Shazam" algorithm for music fingerprinting as a neural network and build a differentiable implementation of it in TensorFlow [48]. By using a gradient-based attack and an objective that is designed to achieve good transferability to black-box models, we create adversarial music that is easily recognizable to a human, while evading detection by a machine. With sufficient perturbations, our adversarial music successfully fools industrial systems, [1] including the AudioTag music recognition service [49], and YouTube's Content ID system[50]. Sample audio can be found here[2].

4.2 What makes copyright detection systems vulnerable to attacks?

Work on adversarial examples has been focused largely on imaging problems, including image classification, object detection, and semantic segmentation [8, 10, 51, 52, 53, 54, 55]. More recently, adversarial examples have been studied for non-vision

[1]Affected parties were notified before the publication of this article.

[2]https://www.cs.umd.edu/ tomg/projects/copyrightattack/

applications such as speech recognition (i.e., speech-to-text) [5, 56, 57, 58]. Attacks on copyright detection systems are different from these applications in a number of important ways that result in increased potential for vulnerability.

First, digital media can be directly uploaded to a server without passing through a microphone or camera. This is drastically different from physical-world attacks, where adversarial perturbations must survive a data measurement process. For example, a perturbation to a stop sign must be effective when viewed through different cameras, resolutions, lighting conditions, viewing angles, motion blurs, and with different post-processing and compression algorithms. While attacks exist that are robust to these nuisance variables [3], this difficulty makes even white-box attacks difficult, leaving some to believe that physical world attacks are not a realistic threat model [59, 60]. In contrast, a manipulated audio file can be uploaded directly to the web without passing it through a microphone that may render perturbations ineffective.

Second, copyright detection is an *open-set* problem, in which systems process media that does not fall into any known class (i.e., doesn't correspond to any protected audio/video). This is different from the closed-set detection problem in which everything is assumed to correspond to a class. For example, a mobile phone application for music identification may solve a closed-set problem; the developers can assume that every uploaded audio clip corresponds to a known song, and when results are uncertain there is no harm in guessing. By contrast, when the same algorithm is used for copyright detection on a server, developers must solve the open-set problem; nearly all uploaded content is not copyright protected and should be labeled as such. In this case, there is harm in "guessing" an ID when results are uncertain, as this may bar users from uploading non-protected material. Copyright detection algorithms must be tuned conservatively to operate in an environment where most content does not get flagged.

Finally, copyright detection systems must handle a deluge of content with different labels despite strong feature similarities. Adversarial attacks are known to succeed easily in an environment where two legitimately different audio/video clips may share strong similarities at the feature level. This has been recognized for the ImageNet classification task [61], where feature overlap between classes (e.g., numerous classes exist for different types of cats/dogs/birds) makes systems highly vulnerable to untargeted attacks in which the attacker perturbs an object from its home class into a different class of high similarity. As a result, the state of the art defenses for untargeted attacks on ImageNet achieve far lower robustness than classifiers for simpler tasks [62, 63]. Copyright detection systems may suffer from a similar problem; they must discern between protected and non-protected content even when there is a strong feature overlap between the two.

4.3 Types of copyright detection systems

Fingerprinting algorithms typically work by extracting an ensemble of feature vectors (also called a “hash” in the case of audio tagging) from source content, and then matching these vectors to a library of known vectors associated with copyrighted material. If there are enough matches between a source sample and a library sample, then the two samples are considered identical. Most audio, image, and video fingerprinting algorithms either train a neural network to extract fingerprint features, or extract hand-crafted features. In the former case, standard adversarial methods lead to immediate susceptibility. In the latter case, feature extractors can often be re-interpreted and implemented as shallow neural networks, and then attacked (we will see an example of this below).

For video fingerprinting, one successful approach by [64] is to use object detectors to identify objects entering/leaving video frames. An extracted hash then consists of features describing the entering/leaving objects, in addition to the temporal relationships between them. While effective at labeling clean video, recent work has shown that object detectors and segmentation engines are easily manipulated to adversarially place/remove objects from frames [54, 65].

Works such as [66] build "robust" fingerprints by training networks on commonly used distortions (such as adding a border, adding noise, or flipping the video), but do not consider adversarial perturbations. While such networks are robust against pre-defined distortions, they will not be robust against white-box (or even black-box) adversarial attacks.

Similarly, recent plagiarism detection systems such as [67] rely on neural networks to generate a fingerprint for a document. While using the deep feature representations of a document as a fingerprint might result in higher accuracy for the plagiarism model, it potentially leaves the system open to adversarial attacks.

Audio fingerprinting might appear to be more secure than the domains described above because practitioners typically rely on hand-crafted features rather than deep neural nets. However, we will see below that even hand-crafted feature extractors are susceptible to attacks.

4.4 Case study: evading audio fingerprinting

We now describe a commonly used audio fingerprinting/detection algorithm and show how one can build a differentiable neural network resembling this algorithm. This model can then be used to mount black-box attacks on real-world systems.

4.4.1 Audio fingerprinting models

An acoustic fingerprint is a feature vector that is useful for quickly locating a sample or finding similar samples in an audio database. Audio fingerprinting plays a central role in detection algorithms such as Content ID. Therefore, in this section, we describe a generic audio fingerprinting model that will ultimately help us generate adversarial examples.

4.4.1.1 Important fingerprinting guidelines from Shazam

Due to the financially sensitive nature of copyright detection, there are very few publicly available fingerprinting models. One of the few widely used publicly known models is from the Shazam team [68]. Shazam is a popular mobile phone app for identifying music. According to the Shazam paper, a good audio fingerprint should have the following properties:

- *Temporally localized*: every fingerprint hash is calculated using audio samples that span a short time interval. This enables hashes to be matched to a short sub-sample of a song.

- *Translation invariant*: fingerprint hashes are (nearly) the same regardless of where in the song a sample starts and ends.

- *Robust*: hashes generated from the original clean database track should be reproducible from a degraded copy of the audio.

4.4.1.2 The handcrafted fingerprinting model

The *spectrogram* of a signal, also called the short-time Fourier transform, is a plot that shows the frequency content (Fourier transform) of the waveform over time. After experimenting with various features for fingerprinting, [68] chose to form hashes from the locations of spectrogram peaks. Spectrogram peaks have nice properties such as robustness in the presence of noise and approximate linear superposability.

In the next subsection, we build a shallow neural network that captures the key ideas of [68], while adding extra layers that help produce transferable adversarial examples. In particular, we add an extra smoothing layer that makes our model difficult to attack and helps us craft strong attacks that can transfer to other black-box models.

4.4.2 Interpreting the fingerprint extractor as a CNN

Here we describe the details of the generic neural network model we use for generating the audio fingerprints. Each layer of the network can be seen as a transformation that is applied to its input. We treat the output representation of our network as the fingerprint of the input audio signal. Ideally, we would like to extract features that can uniquely identify a signal while being independent of the exact start or end time of the sample. Convolutional neural networks maintain the *temporally localized* and *translation invariant* properties mentioned in section 4.4.1.1, and so we model the fingerprinting procedure using fully convolutional neural networks.

The first network layer convolves with a normalized Hann function, which is a filter of the form

$$f_1(n) = \frac{sin^2\left(\frac{\pi n}{N}\right)}{\sum_{i=0}^{N} sin^2\left(\frac{\pi i}{N}\right),} \tag{4.1}$$

where N is the width of the kernel. Convolving with a normalized Hann window smooths the adversarially perturbed audio waveform and the output of this layer is a perturbed but smooth audio sample that is then fingerprinted. This layer removes discontinuities and bad spectral properties that may be introduced into the signal during adversarial optimization and also makes the black-box attacks more efficient by preventing perturbations that do not transfer well to other models.

The next convolutional layer computes the spectrogram (aka Short Term Fourier Transform) of the waveform and converts the audio signal from its original domain to a representation in the frequency domain. This is accomplished by convolving with an ensemble of N Fourier kernels of different frequencies, each with N output channels. This convolution has filters of the form

$$f_2(k,n) = e^{-i2\pi kn/N}, \tag{4.2}$$

where $k \in 0, 1, \cdots, N-1$ is an output channel index and $n \in 0, 1, \cdots, N-1$ is the index of the filter coefficient. After this convolution is computed, we apply $|x|$ on the

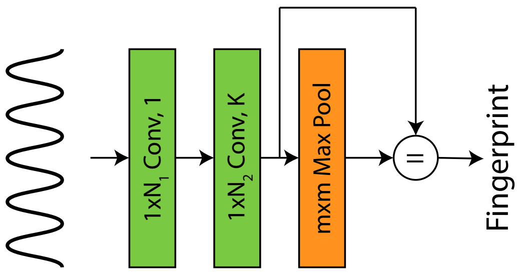


Figure 4.1: An audio fingerprinting model with two convolution layers and a max pooling layer. This model produces binary fingerprints by finding local maxima of the spectrogram of the input signal.

output to get the magnitude of the STFT.

After the convolutional layers, we get a feature representation of the audio signal. We call this feature representation $\phi(x)$, where x is the input signal. This representation is susceptible to noise and a slight perturbation in the audio signal can change it. Furthermore, this representation is very dense which makes it relatively hard to store and search against all audio signals in the database. To address these issues, [68] suggests using the local maxima of the spectrogram as features.

We can find local maxima within our neural net framework by applying a max-pooling function over the feature representation $\phi(x)$. We then find the places where the output of the maxpool equals the original feature representation (i.e., the locations where $\phi(x) = maxpool\left(\phi(x)\right)$). The resulting binary map of local maxima locations is the fingerprint of the signal and can be used to search for a signal against a database of previously processed signals. We will refer to this binary fingerprint as $\psi\left(x\right)$ where x is the input signal. Figure 4.1 depicts the 2-layer convolutional network we use in this work for generating signal fingerprints.

4.4.3 Formulating the adversarial loss function

To craft an adversarial perturbation, we need a differentiable surrogate loss that measures how well an extracted fingerprint matches a reference. The CNN described in section 4.4.2 uses spectrogram peaks to generate fingerprints, but we did not yet specify a loss for quantifying how close two fingerprints are. Once we have such a loss, we can use standard gradient methods to find a perturbation δ that can be added to an audio signal to prevent copyright detection. To ensure the similarity between perturbed and clean audio, we bound the perturbation δ. That is, we enforce $\|\delta\|_p \leq \epsilon$. Here $\|.\|_p$ is the ℓ_p-norm of the perturbation and ϵ is the perturbation budget available to the adversary. In our experiments, we use the ℓ_∞-norm as our measure of perturbation size.

The simplest similarity measure between two binary fingerprints is simply the Hamming distance. Since the fingerprinting model outputs a binary fingerprint $\psi(x)$, we can simply measure the number of local maxima that the signals x and y share by $|\psi(x) \cdot \psi(y)|$. To make a differentiable loss function from this similarity measure, we use

$$J(x, y) = |\phi(x) \cdot \psi(x) \cdot \psi(y)|. \tag{4.3}$$

In the white box case where the fingerprinting system is known, attacks using the loss (4.3) are extremely effective. However, attacks using this loss are extremely brittle and do not transfer well; one can minimize this loss by changing the locations of local maxima in the spectrogram by just one pixel. Such small changes in the spectrogram are unlikely to transfer reliably to black-box industrial systems.

To improve the transferability of our adversarial examples, we propose a robust loss that promotes large movements in the local maxima of the spectrogram. We do this by moving the locations of local maxima in $\phi(x)$ outside of any neighborhood of the local maxima of $\phi(y)$. To efficiently implement this constraint within a neural net framework, we use two separate max-pooling layers, one with a bigger width w_1 (the same width used in fingerprint generation), and the other with a smaller width w_2. If a location in the spectrogram yields output of the w_1 pooling strictly larger than the output of the w_2 pooling[3], we can be sure that there is no spectrogram peak within radius w_2 of that location.

Equation 4.4 describes a loss function that penalizes the local maxima of x that are in the w_2 neighborhood of local maxima of y. This loss function forces the output of the max pooling layers to be different by at least a margin c.

$$\begin{aligned} J(x,y) = \sum_i \Bigg(ReLU \Bigg(c - \Bigg(& \max_{|j| \leq w_1} \phi(i+j;x) \\ & - \max_{|j| \leq w_2} \phi(i+j;x) \Bigg) \Bigg) \cdot \psi(i;y) \Bigg) \end{aligned} \tag{4.4}$$

Finally, we make our loss function differentiable by replacing the maximum operator with the smoothed max function

$$S_\alpha (x_1, x_2, \cdots, x_n) = \frac{\sum_{i=1}^{n} x_i e^{\alpha x_i}}{\sum_{i=1}^{n} e^{\alpha x_i}}, \tag{4.5}$$

where α is a smoothing hyper parameter. As $\alpha \to \infty$, the smoothed max function more

[3]The first max-pooling layer's output is always greater than or equal to the output of the second max-pooling layer.

accurately approximates the exact max function. For simplicity, we chose $\alpha = 1$ for all experiments.

4.4.4 Crafting the evasion attack

We solve the bounded optimization problem

$$\min_{\delta} J(x + \delta, x) \qquad s.t. \|\delta\|_{\infty} \leq \epsilon, \tag{4.6}$$

where x is the benign audio sample, and J is the loss function defined in equation 4.4 with the smoothed max function. Note that unlike common adversarial example generation problems from the literature, our formulation is a minimization problem because of how we defined the objective. We solve (4.6) using projected gradient descent [69] in which each iteration updates the perturbation using Adam [70], and then clips the perturbation to ensure that the ℓ_{∞} constraint is satisfied.

4.4.5 Remix adversarial examples

The optimization problem defined in equation 4.6 tries to create an adversarial example with a fingerprint that does not look like the original signal's fingerprint. While this approach can trick the search algorithm used in copyright detection systems by lowering its confidence, it can result in unnatural sounding perturbations. Alternatively, we can try to enforce the perturbed signal's fingerprint to be similar to a different audio signal. Due to the approximate linear superposability characteristic of the spectrogram peaks, this will make the adversarial example sound more natural and like the target signal audio.

To achieve this goal, we will first introduce a loss function that tries to make two signals look similar rather than different. As described in equation 4.7, such a loss can be obtained by replacing the order of max over big and small neighborhoods in equation 4.4. Note that we will still use the smooth maximum from equation 4.5.

$$\begin{aligned} J_{remix}(x, y) = \sum_i \Bigg(ReLU \Bigg(c - \Bigg(\max_{|j| \le w_2} \phi(i+j; x) \\ - \max_{|j| \le w_1} \phi(i+j; x) \Bigg) \Bigg) \cdot \psi(i; y) \Bigg) \end{aligned} \tag{4.7}$$

Using this loss function, we define the optimization problem in equation 4.8, which not only tries to make the adversarial example different from the original signal x, but also forces similarity to another signal y.

$$\begin{aligned} \min_\delta J(x+\delta, x) + \lambda J_{remix}(x+\delta, y) \\ s.t. \quad \|\delta\|_p \le \epsilon. \end{aligned} \tag{4.8}$$

Here λ is a scale parameter that controls how much we enforce the similarity between the fingerprints of $x + \delta$ and y. We call adversarial examples generated using equation 4.8 "remix" adversarial examples as they sound more like a remix, and refer to examples generated using equation 4.6 as default adversarial examples. While a successful attack's adversarial perturbation may be larger in the case of remix adversarial examples (due to the additional term in the objective function), the perturbation sounds more natural.

4.5 Evaluating transfer attacks on industrial systems

We test the effectiveness of our black-box attacks on two real-world audio search/copyright detection systems. The inner workings of both systems are proprietary, and therefore it is necessary to attack these systems with black-box transfer attacks. Both systems claim to be robust against noise and other input signal distortions.

We test our system on a dataset containing the top billboard songs from the past 10 years. We extract a 30-second fragment of these songs and craft both our default and remix adversarial examples for them. Although both types of adversarial examples can dodge detection, they have very different characteristics. The default adversarial examples (equation 4.6) work by removing identifiable frequencies from the original signal, while the remix adversarial examples (equation 4.8) work by introducing new frequencies to the signal that will confuse the real-world systems.

Sample audio files can be found here[4].

4.5.1 White-box attack results

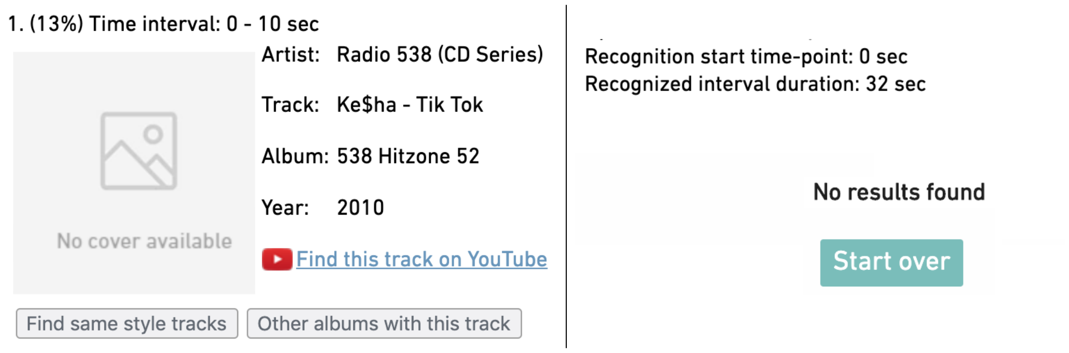
Before evaluating black-box transfer attacks against real-world systems, we evaluate the effectiveness of a white-box attack against our own proposed model. Doing so will allow us to have a baseline of how effective an adversarial example can be if the details of a model are ever released or leaked.

To create white-box attacks against our model, we use the loss function defined in equation 4.3. By optimizing this function, we can remove almost all of the fingerprints

[4]https://www.cs.umd.edu/ tomg/projects/copyrightattack/

Percentage of removed hashes	90%	95%	99%
Perturbation norm (ℓ_∞)	0.012	0.023	0.038
Perturbation norm (ℓ_2)	0.004	0.005	0.006

Table 4.1: Norms of the perturbations for white-box attacks. Before computing the norms, we have normalized the signals to have samples that lie in $[0, 1]$.



1. (13%) Time interval: 0 - 10 sec

Artist: Radio 538 (CD Series)

Track: Ke$ha - Tik Tok

Album: 538 Hitzone 52

Year: 2010

Recognition start time-point: 0 sec

Recognized interval duration: 32 sec

No results found

Figure 4.2: AudioTag can identify the benign audio signals, but fails to detect adversarial examples.

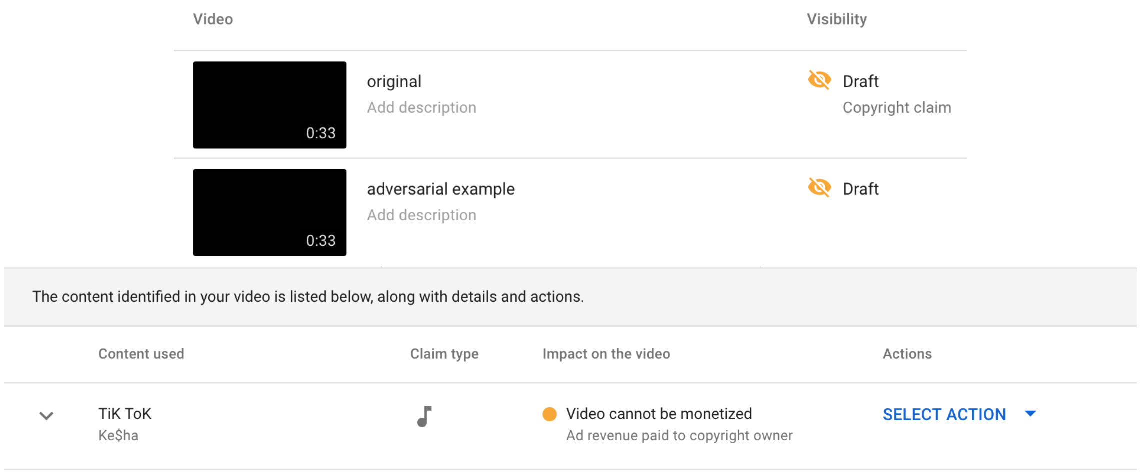
identified by our model with perturbations that are unnoticeable by humans. Table 4.1 shows the norms of the perturbations required to remove 90%, 95%, and 99% of the fingerprint hashes.

4.5.2 Transfer attacks on AudioTag

AudioTag[5] is a free music recognition service with millions of songs in its database. When a user uploads a short audio fragment on this website, AudioTag compares the audio fragment against a database of songs and identifies what song this audio fragment belongs to. AudioTag claims to be “robust to sound distortions, noises and even speed variation, and will therefore recognize songs even in low quality audio recordings”.[6] Therefore, one would expect that low-amplitude non-adversarial noise should not affect

[5]https://audiotag.info/

[6]https://audiotag.info/faq


Video	Visibility
original Add description 0:33	Draft Copyright claim
adversarial example Add description 0:33	Draft

The content identified in your video is listed below, along with details and actions.

Content used	Claim type	Impact on the video	Actions
TiK ToK Ke$ha	♪	● Video cannot be monetized Ad revenue paid to copyright owner	SELECT ACTION

Figure 4.3: YouTube can successfully identify the benign/original audio signal while it fails to detect the adversarial examples.

this system.

As shown in Figure 4.2, AudioTag can accurately detect the songs corresponding to the benign signal. However, the system fails to detect both the default and remix adversarial examples built for them. During our experiments with AudioTag, we realized that this system is relatively sensitive to our proposed attacks and it can be fooled with relatively small perturbation budgets. Qualitatively, the magnitude of the noise required to fool this system is small and it is not easily noticeable by humans. Based on this observation, we suspect that the architecture of the fingerprinting model used in AudioTag may have similarities to our surrogate model in section 4.4.2.

Table 4.2 shows the ℓ_∞ and ℓ_2 norms of the perturbations required to fool AudioTag on 90% of the songs in our dataset. We also verified AudioTag's claim of being robust to input distortions by applying random perturbations to the audio recordings. To fool AudioTag with random noise, the magnitude (ℓ_∞) of the noise must be roughly 4 times larger than the noise we craft using equation 4.6.

Target model	AudioTag			YouTube		
Type of perturbation	default	remix	random	default	remix	random
Perturbation norm (ℓ_∞)	0.03	0.03	0.12	0.10	0.10	0.32
Perturbation norm (ℓ_2)	0.02	0.02	0.06	0.07	0.08	0.19

Table 4.2: Norms of the perturbations in adversarial examples that can evade each real-world system. Before computing the norms, we have normalized the signals to $[0, 1]$.

4.5.3 YouTube

YouTube[7] is a video sharing website that allows users to upload their video files. YouTube has developed a system called "Content ID[8]" to automatically tag user-uploaded content that contains copyrighted material. Using this system, copyright owners can submit their content and have YouTube scan uploaded videos against it.

As shown in the screenshot in Figure 4.3, YouTube Content ID can successfully identify the benign songs we use in our experiment. At the time of writing this chapter both our default and remix attacks successfully evade Content ID and go undetected. However, YouTube Content ID is significantly more robust to our attacks than AudioTag. To fool Content ID, we had to use a larger value for ϵ. This makes perturbations quite noticeable, although songs are still immediately recognizable by a human. Furthermore, a perturbation with non-adversarial random noise must have an ℓ_∞ norm 3 times larger than our adversarial perturbations to successfully avoid being detected.

We repeated our experiments with identical hyper-parameters on the songs from our dataset. Table 4.2 shows the ℓ_∞ norms (i.e., the parameter ϵ) and ℓ_2 norms of the perturbations required to fool YouTube on 67% of the songs in our dataset. Furthermore,

[7]https://www.youtube.com/

[8]https://support.google.com/youtube/answer/2797370?hl=en

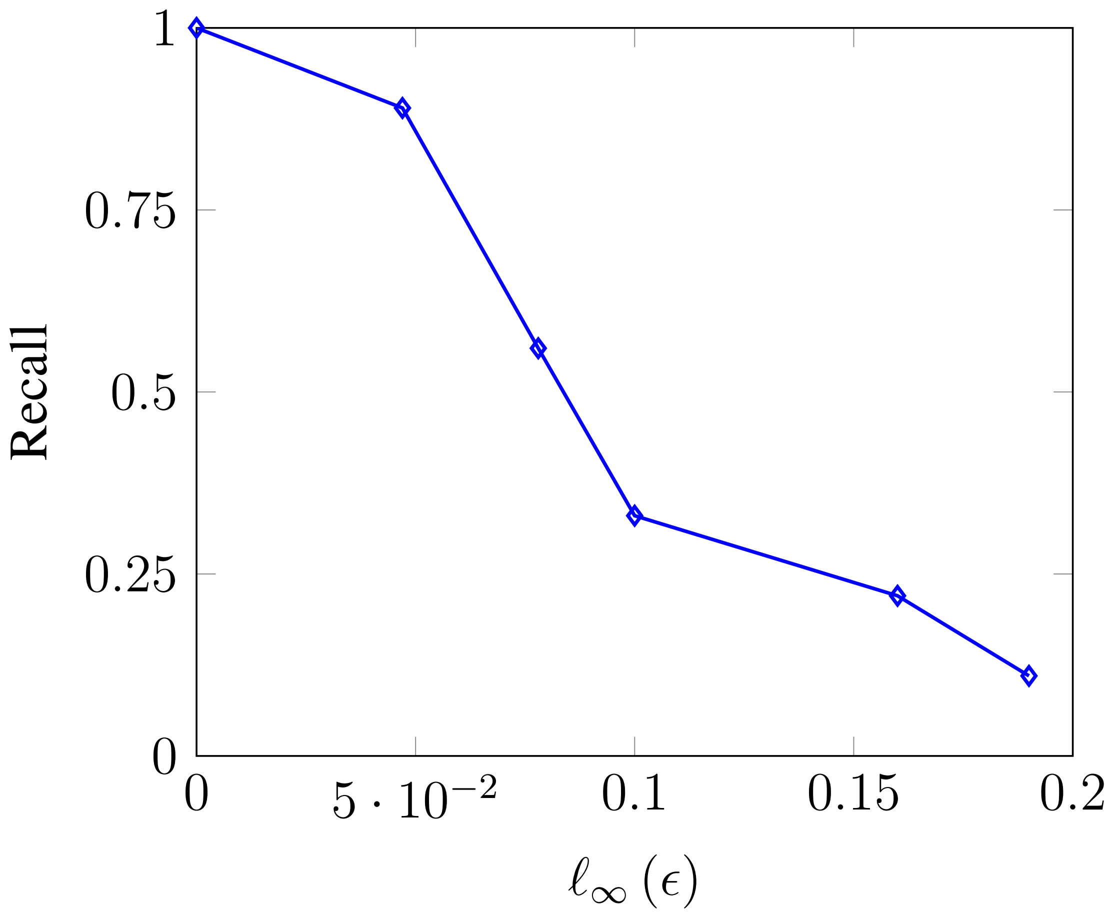


Figure 4.4: YouTube's copyright detection recall against the magnitude of noise on top Billboard songs dataset

figure 4.4 shows the recall of YouTube's copyright detection tool on our dataset for different magnitudes of perturbations.

4.6 Conclusion

Copyright detection systems are an important category of machine learning methods, but the robustness of these systems to adversarial attacks has not been addressed yet by the machine learning community. We discussed the vulnerability of copyright detection systems, and explain how different kinds of systems may be vulnerable to attacks using known methods. As proof of concept, we build a simple song identification method using neural network primitives and attack it using well-known gradient methods. Surprisingly, attacks on this model transfer well to online systems.

The implementations used in this study are far from optimal, and we expect that attacks can be strengthened using sharper technical tools, including perturbation types that are less perceptible to the human ear. Furthermore, we are doing transfer attacks using fairly rudimentary surrogate models that rely on hand-crafted features, while commercial systems likely rely on full trainable neural nets.

Our goal here is not to facilitate copyright evasion, but rather to raise awareness of the threats posed by adversarial examples in this space, and to highlight the importance of hardening copyright detection and content control systems to attack. A number of defenses already exist that can be utilized for this purpose, including adversarial training.

Chapter 5: Adversarially robust transfer learning

5.1 Introduction

Deep neural networks achieve human-like accuracy on a range of tasks when sufficient training data and computing power is available. However, when large datasets are unavailable for training, or pracitioners require a low-cost training strategy, *transfer learning* methods are often used. This process starts with a source network (pre-trained on a task for which large datasets are available), which is then re-purposed to act on the target problem, usually with minimal re-training on a small dataset [71, 72].

While transfer learning greatly accelerates the training pipeline and reduces data requirements in the target domain, it does not address the important issue of model *robustness*. It is well-known that naturally trained models often completely fail under adversarial inputs [7, 8]. As a result, researchers and practitioners often resort to *adversarial training*, in which adversarial examples are crafted on-the-fly during network training and injected into the training set. This process greatly exacerbates the problems that transfer learning seeks to avoid. The high cost of creating adversarial examples increases training time (often by an order of magnitude or more). Furthermore, robustness is known to suffer when training on a small dataset [11]. To make things worse, high-capacity models are often needed to achieve good robustness [2, 9, 62], but these models may over-fit badly on small datasets.

Contributions

The purpose of this work is to study the adversarial robustness of models produced by transfer learning. We begin by observing that robust networks contain *robust feature extractors*, which are resistant to adversarial perturbations in different domains. Such robust features can be used as a basis for semi-supervised transfer learning, which only requires re-training the last layer of a network. To demonstrate the power of robust transfer learning, we transfer a robust ImageNet source model onto the CIFAR domain, achieving both high accuracy and robustness in the new domain without adversarial training. We use visualization methods to explore properties of robust feature extractors. Then, we consider the case of transfer of learning by "fine-tuning." In this case, the source network is re-trained end-to-end using a small number of epochs on the target domain. Unfortunately, this end-to-end process does not always retain the robustness of the source domain; the network "forgets" the robust feature representations learned on the source task. To address this problem, we use recently proposed lifelong learning methods that prevent the network from forgetting the robustness it once learned. Using our proposed methods, we construct robust models that generalize well. In particular, we improve the generalization of a robust CIFAR-100 model by roughly 2% while preserving its robustness.

5.2 Background

Adversarial examples fall within the category of evasion attacks—test-time attacks in which a perturbation is added to a natural image before inference. Adversarial attacks are most often crafted using a differentiable loss function that measures the performance of a classifier on a chosen image. In the case of norm-constrained attacks (which form the basis of most standard benchmark problems), the adversary solves

$$\max_{\delta} \quad l(x+\delta, y, \theta) \qquad \text{s.t.} \quad \|\delta\|_p \leq \epsilon, \tag{5.1}$$

where θ are the (already trained and frozen) parameters of classifier $c(x,\theta) \rightarrow \hat{y}$ that maps an image to a class, l is the proxy loss used for classification (often cross-entropy), δ is the image perturbation, (x, y) is the natural image and its true class, and $||.||_p$ is some ℓ_p-norm[1]. The optimization problem in Eq. 5.1 aims to find a bounded perturbation that maximizes the cross-entropy loss given the correct label. There are many variants of this process, including DeepFool [52], L-BFGS [8], and CW [73].

Many researchers have studied methods for building a robust network which have been later shown to be ineffective when attacked with stronger adversaries [74]. Adversarial training [8] is one of the defenses that was not broken by [74]. While adversarial training using a weak adversary such as the FGSM attack [51] can be broken even by single step attacks which add a simple random step prior to the FGSM step [75], adversarial training using a strong attack has successfully improved robustness. [9]

[1]By default we will use the ℓ_∞-norm in this chapter.

showed that a PGD attack (which is a BIM attack [2] with an initial random step and projection) is a strong enough attack to achieve promising adversarial training results. We will refer to this training method as *PGD adversarial training*. PGD adversarial training achieves good robustness on bounded attacks for MNIST [76] and acceptable robustness on CIFAR-10 [77] classifiers.

[78] show that adversarial training with strong PGD adversaries has many benefits in addition to robustness. They also state that while adversarial training may improve generalization in regimes where training data is limited (especially on MNIST), it may be at odds with generalization in regimes where data is available. This trade-off was also recently studied by [79], [80], and [81].

While, to the best of our knowlegde, the transferability of robustness has not been studied in depth, [82] studied the case of adversarially training models that were pre-trained on different domains. Our work is fundamentally different in that we seek to transfer robustness without resorting to costly and data-hungry adversarial training. We train the target model on natural examples only, which allows us to directly study how well robustness transfers. Additionally, this allows us to have better generalization and achieve higher accuracy on validation examples. While as [82] state, fine-tuning on adversarial examples built for the target domain can improve robustness of relatively large datasets such as CIFAR-10 and CIFAR-100 compared to adversarial training from scratch on the target domain, we show that in the regimes of limited data (where transfer learning is more common), adversarially robust transfer learning can lead to better results measured in terms of both robustness and clean validation accuracy.

Table 5.1: Accuracy and robustness of natural and adversarially trained models on CIFAR-10+ and CIFAR-100+. The "+" sign denotes standard data augmentation.

Dataset	model	validation accuracy	accuracy against PGD-20
CIFAR-10+	natural	95.01%	0.00%
	robust	87.25%	45.84%
CIFAR-100+	natural	78.84%	0.00%
	robust	59.87%	22.76%

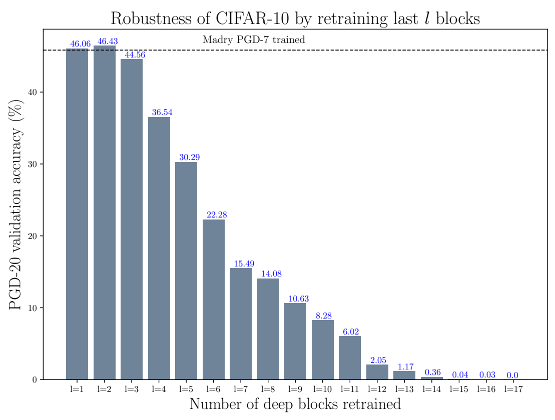
5.3 The robustness of deep features

In this section, we explore the robustness of different network layers, and demonstrate that robust networks rely on robust deep features. To do so, we start from robust classifiers ($c(\theta_r)$) for the CIFAR-100 and CIFAR-10 datasets [77], and update θ by training on natural examples. In each experiment, we re-initialize the last k layers/blocks of the network, and re-train just those layers. We start by re-initializing just the last layer, then the last two, and so on until we re-initialize all the layers.

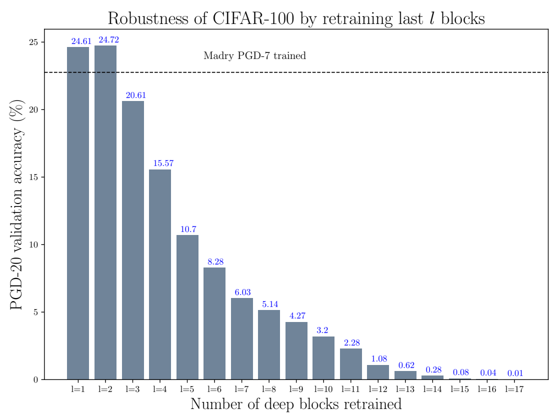
We use the adversarially trained Wide-ResNet 32-10 [83] for CIFAR-10 from [9] as our robust model for CIFAR-10. We also adversarially train our own robust classifier for CIFAR-100 using the code from [9]. To keep things consistent, we use the same hyper-parameters used by [9] for adversarially training CIFAR-10 to adversarially train the CIFAR-100 model.[2] The performance of the CIFAR-10 and CIFAR-100 models on natural and adversarial examples are summarized in Table 5.1. To measure robustness, we evaluate the models on adversarial examples built using PGD attacks.

We break the WRN 32-10 model into 17 blocks, which are depicted in Fig. 5.2. In each experiment, we first re-initialize the k deepest blocks (blocks 1 through k) and then

[2]We adv. train the WRN 32-10 on CIFAR-100 using a 7-step ℓ_∞ PGD attack with step-size=2 and $\epsilon = 8$. We train for 80,000 iterations with a batch-size of 128.



(a) CIFAR-10 PGD-20 accuracy



(b) CIFAR-100 PGD-20 accuracy

Figure 5.1: Robustness is preserved when we retrain only the deepest block(s) of robust CIFAR-10 and CIFAR-100 models using natural examples. The vertical axis is the accuracy on PGD-20 generated adversarial examples (*i.e*let@tokeneonedotrobustness) after re-training deep layers. The robustness of the adversarially trained models if all layers are frozen are shown with dashed lines.

train the parameters of those blocks on natural images[3]. We train for 20,000 iterations using Momentum SGD and a learning rate of 0.001. We then incrementally unfreeze and train more blocks. For each experiment, we evaluate the newly trained model's accuracy on validation adversarial examples built with a 20-step PGD ℓ_∞ attack with $\epsilon = 8$.

Fig. 5.1 shows that robustness does not drop if only the final layers of the networks are re-trained on natural examples. In fact, there is a slight increase in robustness compared to the baseline PGD-7 adversarially trained models when we just retrain the last batch-normalization block and fully connected block. As we unfreeze and train more blocks, the network's robustness suddenly drops. This leads us to believe that a hardened network's robustness is mainly due to robust deep feature representations and robustness is preserved if we re-train on top of deep features.

Now that we have identified feature extractors as a source of robustness, it is natural

[3]In this experiment, we use standard data augmentation techniques.

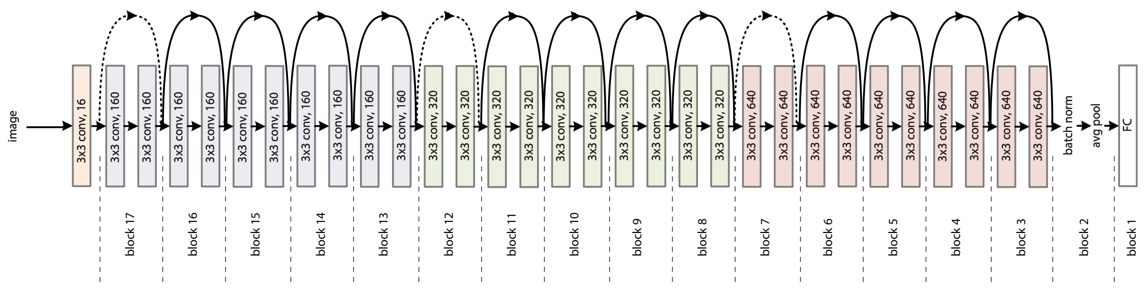


Figure 5.2: Wide Resnet 32-10 and the blocks used for freezing/retraining

to investigate whether robustness is preserved when transfer learning using robust feature extractors. We will study two different approaches for transferring robustness across datasets: one in which only the last layer is re-trained, and one with end-to-end re-training.

5.4 Transfer learning: Recycling feature extractors

We study how robustness transfers when the feature extractor layers of the source network are frozen, and we retrain only the last fully connected layer (*i.e*let@tokeneonedotthe classification layer) for the new task. Formally, the transfer learning objective is:

$$\min_{w} \quad l(z(x, \theta^*), y, w) \tag{5.2}$$

where z is the deep feature extractor function with pre-trained and now "frozen" parameters θ^*, and w represents the trainable parameters of the last fully connected layer. To investigate how well robustness transfers, we use two source models: one that is hardened by adversarial training and another that is naturally trained.

We use models trained on CIFAR-100 as source models and perform transfer

Table 5.2: Transfer learning by freezing the feature extractor layers.

Source Dataset	Target Dataset	Source Model	val.	PGD-20
CIFAR-100	CIFAR-10	natural robust	83.05% 72.05%	0.00% 17.70%
CIFAR-100 (50% of classes)	CIFAR-100 (other 50% of classes)	natural robust	71.44% 58.48%	0.00% 15.86%
CIFAR-100 (50% of classes)	CIFAR-100 (same 50% of classes)	natural robust	80.20% 64.96%	0.00% 25.16%

learning from CIFAR-100 to CIFAR-10. The results are summarized in Table 5.2. Compared to adversarial/natural training the target model, transferring from a source model seems to result in a drop in natural accuracy (compare first row of Table 5.1 to the first row of Table 5.2). This difference is wider when the source and target data distributions are dissimilar [71].

To evaluate our method on two datasets with more similar attributes, we randomly partition CIFAR-100 into two disjoint subsets where each subset contains images corresponding to 50 classes. Table 5.2 shows the accuracy of transferring from one of the disjoint sets to the other (second row) and to the same set (third row). We can compare results of transfer learning with adversarial training on CIFAR-100 by averaging the results in the second and third rows of Table 5.2 to get the accuracy across all 100 classes of CIFAR-100.[4] By doing so, we see that the accuracy of the transferred classifier matches that of the adversarially trained one, even though no adversarial training took place in the target domain.

We make the following observations from the transfer-learning results in Table 5.2.

[4]The robust CIFAR-100 classifier has 59.87% validation accuracy and 22.76% accuracy on PGD-20 adversarial examples. The average validation accuracy of the two half-CIFAR-100 classifiers on validation examples is $\frac{64.96\%+58.48\%}{2} = 61.72\%$ while the average robustness is $\frac{25.16\%+15.86\%}{2} = 20.51\%$.

1) *robustness transfers:* when the source model used for transfer learning is robust, the target model is also robust (although less so than the source), 2) *robustness transfers between models that are more similar:* If the source and target models are trained on datasets which have similar distributions (and number of classes), robustness transfers better, and 3) *validation accuracy drops if we transfer from robust models in comparison to naturally trained source models:* if the source model is naturally trained, the natural validation accuracy is better, although the target model is then vulnerable to adversarial perturbations.

5.4.1 Transfer Learning with ImageNet models

Transfer learning using models trained on ImageNet [61] as the source is a common practice in industry because ImageNet feature extractors are powerful and expressive. In this section we evaluate how well robustness transfers from these models.

5.4.1.1 Transfer learning using ImageNet

Starting from both a natural and robust ImageNet model, we perform the same set of experiments we did in section 5.4. Robust ImageNet models do not withstand untargeted ℓ_∞ attacks using as large an ϵ as those that can be used for simpler datasets like CIFAR. Following the method [62], we "free train" a robust ResNet-50 on ImageNet using replay hyper-parameter $m = 4$. The hardened ImageNet classifier withstands attacks bounded by $\epsilon = 5$. Our robust ImageNet achieves 59.05% top-1 accuracy and roughly 27% accuracy against PGD-20 ℓ_∞ $\epsilon = 5$ attacks on validation examples. We experiment with using this

Table 5.3: Transfer learning from ImageNet.

Architecture and Source Dataset	Target Dataset	Source Model	val.	PGD-20
ResNet-50 ImageNet	CIFAR-10+	natural	90.49%	0.01%
		robust ($\epsilon = 5$)	88.33%	22.66%
	CIFAR-100+	natural	72.84%	0.05%
		robust ($\epsilon = 5$)	68.88%	15.21%
Robust u-ResNet-50 ($\epsilon = 5$) for CIFAR-10+			82.00%	53.11%
Robust u-ResNet-50 ($\epsilon = 5$) for CIFAR-100+			59.90%	29.54%

robust ImageNet model and a conventionally trained ResNet-50 ImageNet model as the source models.

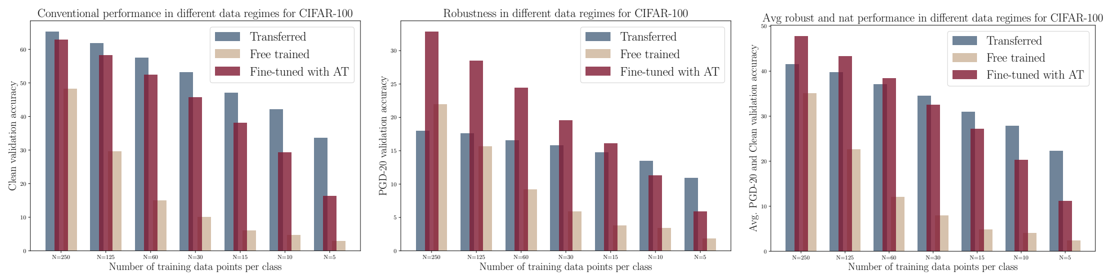
Using the ImageNet source models, we train CIFAR classifiers by retraining the last layer on natural CIFAR examples. We up-sample the 32×32-dimensional CIFAR images to 224×224 before feeding them into the ResNet-50 source models that are trained on ImageNet. For evaluation purposes, we also train robust ResNet-50 models from scratch using [62] for the CIFAR models. To ensure that the transfer learning models and the end-to-end trained robust models have the same capacity and dimensionality, we first upsample the CIFAR images before feeding them to the ResNet-50 model. To distinguish between the common case of training ResNet models on CIFAR images that are 32×32-dimensional, we call our models that are trained on the upsampled CIFAR datasets the upsample-first ResNets or "*u-ResNets*".

Table 5.3 illustrates that using a robust ImageNet model as a source results in high validation accuracy for the transferred CIFAR target models. Also, given that the ImageNet classifier by itself is 27% robust, the CIFAR-10 model maintains the majority of that 27% robustness. When we compare the end-to-end hardened classifiers (robust u-ResNets) with the transferred classifier, we can see that while the robustness is less for the transferred case, transferred models result in considerably better performance on clean validation examples.

5.4.2 Low-data regime

As touched on before, transfer learning is more common in situations where the number of training points in the target domain is limited. Up until now, as a proof of concept, we have illustrated the majority of our experiments on the CIFAR target domains where we have many training points per-class. [82] show that starting from a pre-trained robust ImageNet model and fine-tuning on adversarial examples of the CIFAR domain can improve robustness beyond that of simply adversarial training CIFAR. Here, we illustrate the effect of training data size on robustness and natural performance by running various experiments on subsets of CIFAR-100 where we vary the number of training points per-class (N).

We compare three different hardening methods: (1) Free-training/adversarial training the target domain [62]; (2) fine-tuning using adversarial examples of the target task starting from the Free-4 robust ImageNet model similar to [82]; and (3) training a fully connected layer on top of the frozen feature extractors of the Free-4 robust ImageNet model using natural examples from the target task. For comparing the three different approaches, we look at three metrics: (a) clean validation accuracy; (b) robustness against PGD-20 validation adversarial examples; and (c) average of robustness and clean performance (((a)+(b))/2.) The results are summarized in Fig. 5.3. In the regimes where transfer learning is more common, adversarially robust transfer learning results



(a) Clean validation (b) Adversarial validation (c) Average validation

Figure 5.3: When the number of training data per-class is very limited (right bars), adversarially robust transfer learning [Transferred] is better in all metrics. However, as the number of training data increases (left bars), fine-tuning with adversarial examples of the target domain [Fine-tuned with AT] results in more robustness. Adversarially robust transfer learning always results in models that work better on natural examples and is $3\times$ faster than fine-tuning with adversarial examples of the target domain. Using a pre-trained robust ImageNet improves both robustness and generalization.

in the best overall performance. Adversarially/Free training the target domain results in less robustness and validation accuracy compared to fine-tuning which highlights the importance of pre-training [82]. Note that in terms of computational resources required, the cost of fine-tuning on adversarial examples of the target domain is about $k\times$ our method since it requires generation of adversarial examples using k-step PGD attacks (we set $k = 3$).

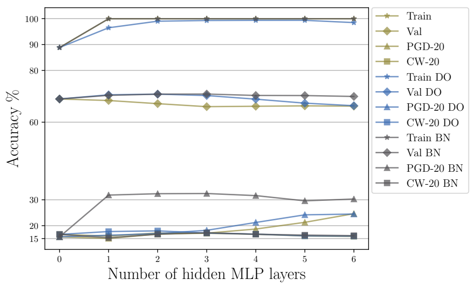
5.4.2.1 Training deeper networks on top of robust feature extractors

The basic transfer learning setting of section 5.4.1.1 only re-trains one layer for the new task. In section 5.4.1.1, when we transferred from the robust ImageNet to CIFAR-100, the natural *training* accuracy was 88.84%. Given the small number of trainable parameters left for the network ($\approx 2048\times100$) and the fixed feature extractor, the network was not capable of completely fitting the training data. This means that there is potential to improve natural accuracy by learning more complex non-linear features and increasing the number of trainable parameters.

To increase representation capacity and the number of trainable parameters, instead of training a 1-layer network on top of the feature extractor, we train a multi-layer perceptron (MLP) network on top of the robust feature extractor. To keep things simple and prevent bottle-necking, every hidden layer we add has 2048 neurons. We plot the training and validation accuracies on the natural examples and the robustness (*i.e*let@tokeneonedotPGD-20 validation accuracy) in Fig. 5.4 for various numbers of hidden layers. As can be seen, adding one layer is enough to achieve 100% training accuracy. However, doing so does not result in an increase in validation accuracy. To the contrary, adding more layers can result in a slight drop in validation accuracy due to overfitting. As illustrated, we can improve generalization using simple but effective methods such as dropout [84] (with probability 0.25) and batch-normalization [85].

However, the most interesting behavior we observe in this experiment is that, as we increase the number of hidden layers, the robustness to PGD-20 attacks improves. Note, this seems to happen even when we transfer from a naturally trained ImageNet model, which leads us to suspect that this behavior may be an artifact of vanishing gradients for adversary as the softmax loss saturates when the data is fit perfectly [74]. Therefore, for this case we change our robustness measure and use the CW attack [73] which will encounter fewer numerical issues because its loss function does not have a softmax component and does not saturate. Attacking the model from the natural source with CW-20 completely breaks the model and achieves 0% robustness. Most interestingly, attacking the model transferred from a robust source using the CW objective maintains robustness even when the number of hidden layers increases.

Figure 5.4: Training an MLP for CIFAR-100 on top of the robust feature extractors from ImageNet. The x-axis corresponds to the number of hidden layers (0 is a linear classifier and corresponds to experiments in section 5.4.1.1). Robustness stems from robust feature extractors. Adding more layers on top of this extractor does not hurt robustness. Interestingly, simply adding more layers does not improve the validation accuracy and just results in more overfitting (*i.e*let@tokeneonedottraining accuracy becomes 100%). We can slightly improve generalization using batch norm (BN) and dropout (DO).



5.5 Analysis: Robust feature extractors are filters

Our experiments suggest that the robustness of neural networks arises in large part from the presence of robust feature extractors. We have used this observation to transfer both robustness and accuracy between domains using transfer-learning. However, we have not yet fully delved into what it means to have a robust feature extractor. Through visualizations, [78] studied how adversarial training causes the image gradients of neural networks to exhibit meaningful generative behavior. In other words, adversarial perturbations on hardened networks "look like" the class into which the image is perturbed. Given that optimization-based attacks build adversarial examples using the image gradient, we also visualize the image gradients of our transferred models to see if they exhibit the same generative behavior as adversarially trained nets.

Fig. 5.5 plots the gradient of the loss w.r.t. the input image for models obtained by re-training only the last layer, and also for the case where we train MLPs on top of a robust feature extractor. The gradients for the transfer-learned models with a robust source are

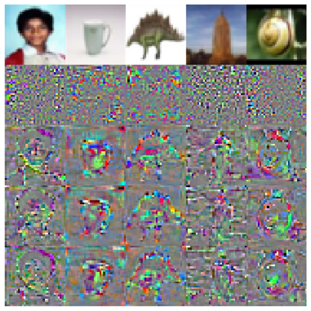


Figure 5.5: Gradients of the loss w.r.t to input images for the CIFAR-100 transfer learning experiments of sections 5.4.1.1 & 5.4.2.1. The top row contains sample CIFAR-100 images. Other rows contain image gradients of the model loss. The second row is for a model transferred from a naturally trained ImageNet source. Rows 3-5 are for models transferred from a robust ImageNet source. These rows correspond to an MLP with 0 (row 3), 1 (row 4), and 2 (row 5) hidden layers on top of the robust feature extractor. The gradients in the last three rows all show interpretable generative behavior.

interpretable and "look like" the adversarial object class, while the gradients of models transferred from a natural source do not. This interpretatbility comes despite the fact that the source model was hardened against attacks on one dataset, and the transferred model is being tested on object classes from another. Also, we see that adding more layers on top of the feature extractor, which often leads to over-fitting, does not make gradients less interpretable. This latter observation is consistent with our observation that added layers preserve robustness(Fig. 5.4).

These observations, together with the success of robust transfer learning, leads us to speculate that a robust model's feature extractors act as a "filter" that ignores irrelevant parts of the image.

5.6 End-to-end training without forgetting

As discussed in section 5.4, transfer learning can preserve robustness of the source model. However, it comes at the cost of decreased validation accuracy on natural examples. The trade-off between generalization and robustness is the subject of recent research [78, 79, 81]. In this section, we intend to decrease the performance gap (on

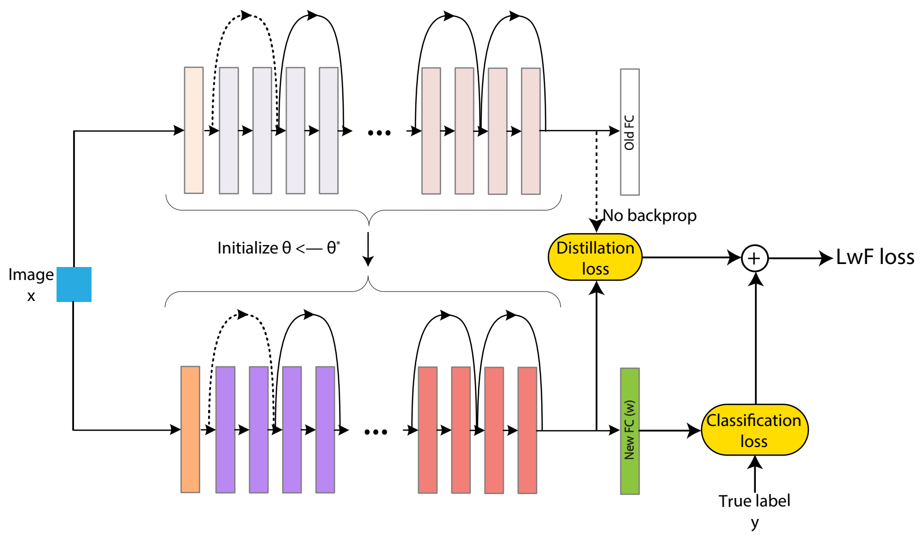


Figure 5.6: Our LwF loss has a term that enforces the similarity of feature representations (*i.e*let@tokeneonedotpenultimate layer activations) between the source model and the fine-tuned model.

natural images) between target models that are trained using natural and robust feature extractors. To do so, we need to fine tune the feature extractor parameters θ. Ideally, we should learn to perform well on the target dataset without catastrophically forgetting the robustness of the source model. To achieve this, we utilize lifelong learning methods.

Learning without Forgetting (LwF) [86] is a method for overcoming catastrophic forgetting. The method is based on distillation. In this framework, we train the target model with a loss that includes a distillation term from the previous model.

$$\min_{w,\theta} \quad l(z(x,\theta), y, w) + \lambda_d \cdot d(z(x,\theta), z_0(x, \theta_r^*)) \tag{5.3}$$

where λ_d is the feature representation similarity penalty, and d is some distance metric between the robust model's feature representations $z_0(x, \theta_r^*)$ and the current model's feature representations $z(x,\theta)$. Unlike the original LwF paper that used a distilled loss from [87], we simply choose d to be the ℓ_2-norm. Our loss is designed to make the feature representations of the source and target network similar, thus preserving the robust feature representations (Fig. 5.6). Ideally, $z(x,\theta) \approx z(x,\theta_r^*)$. To speed up training, given robust feature extractor parameters θ_r^*, we store $z_0(x,\theta_r^*)$ for the images of the target task and load this from memory (*i.e*let@tokeneonedotoffline) instead of performing a forward pass through the robust source network online. Therefore, in the experiments related to LwF, we do not train with data augmentation because we have not pre-computed $z(x_a,\theta_r^*)$, where x_a is the augmented image. Empirically we verified that $d(z(x,\theta_r^*), z(x_a,\theta_r^*))$ was not negligible[5].

To improve performance, we follow a warm-start scheme and only train the fully connected parameters w early in training. We then cut the learning rate and continue fine tuning both feature extractors (θ) and w. In our experiments, we use a learning rate of 0.001, and the warm-start makes up half of the total training iterations. Starting from the pre-trained source model, we train for a total of 20,000 iterations with batch-size 128. The results with an adversarially trained CIFAR-100 model as source and CIFAR-10 as target are summarized in Table 5.4.

As can be seen, having a LwF-type regularizer helps in maintaining robustness and also results in a considerable increase in validation accuracy. The trade-off between robustness and generalization can be controlled by the choice of λ_d. It seems that for some choices of λ_d such as 0.1, robustness also increases. However, in hindsight, the increase in accuracy on PGD-20 adversarial examples is not solely due to improvement in robustness. It is due to the fact that the validation accuracy has increased and we have a better classifier overall. For easier comparisons, we have provided the transfer results

[5]The analysis is in the supplementary.

Table 5.4: Distilling robust features using learning without forgetting. The bottom rows show results from transfer learning with a frozen feature extractor. The '+' sign refers to using augmentation.

Source $\rightarrow$ Target Dataset	Source Model	λ_d	val.	PGD-20
CIFAR-100+ $\rightarrow$ CIFAR-10	robust	1e-7	89.07%	0.61%
		0.001	86.15%	4.70%
		0.0025	81.90%	15.42%
		0.005	**79.35%**	**17.61%**
		0.01	77.73%	17.55%
		0.1	73.39%	18.62%
CIFAR-100+ $\rightarrow$ CIFAR-10+	natural	NA	83.05%	0.00%
	robust	NA	72.05%	17.70%

without LwF at the bottom of Table 5.4. Note that using LwF, we can keep the robustness of the source model and also achieve clean validation accuracy comparable to a model that uses naturally trained feature extractors. In the supplementary, we show that similar conclusions can be drawn for the split CIFAR-100 task.

Decreasing generalization gap of adversarially trained networks

We demonstrated in our transfer experiments that using our LwF-type loss, can help decrease the generalization gap while preserving robustness. In this section, we assume that the source domain is the adversarial example domain of a dataset and the target domain is the clean example domain of the same dataset. This experiment can be seen as applying transfer learning from the adversarial example domain to the natural example domain while preventing forgetting the adversarial domain.

In the case where the source and target datasets are the same (Transferring from a robust CIFAR-100 model to CIFAR-100), by applying our LwF-type loss, we can improve the generalization of robust models. Our results are summarized in Table 5.5.

Table 5.5: Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-100. The '+' sign refers to using augmentation.

Source $\rightarrow$ Target Dataset	Source Model	λ_d	val.	PGD-20
CIFAR-100+ $\rightarrow$ CIFAR-100	robust	1e-5	61.53%	21.83%
		5e-5	**61.71%**	**23.44%**
		1e-4	61.38%	23.95%
		0.001	60.17%	24.31%
		0.01	59.87%	24.10%
CIFAR-100+	robust	NA	59.87%	22.76%

5.7 Conclusion

We identified the feature extractors of adversarially trained models as a source of robustness, and use this observation to transfer robustness to new problems domains without adversarial training. While transferring from a natural model can achieve higher validation accuracy in comparison to transferring from a robust model, we can close the gap and maintain the initial transferred robustness by borrowing ideas from the lifelong learning literature. The success of this methods suggests that a robust feature extractor is effectively a filter that sifts out relevant components of an image that are needed to assign class labels. We hope that the insights from this study enable practitioners to build robust models in situations with limited labeled training data, or when the cost and complexity of adversarial training from scratch is untenable.

5.8 Experiment details

5.8.1 LWF-based experiments

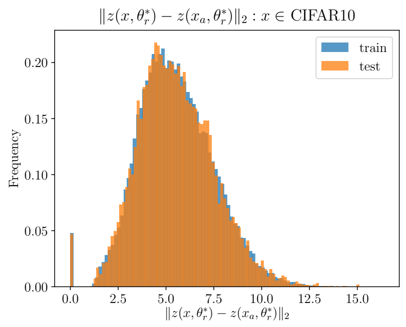
In our LWF-based experiments, we use a batch-size of 128, a fixed learning-rate of 1e-2m, and fine-tune for an additional 20,000 iterations. The first 10,000 iterations are used for warm-start; during which we only update the final fully connected layer's weights. During the remaining 10,000 iterations, we update all of the weights but do not update the batch-normalization parameters.

5.8.2 ImageNet to CIFAR experiments

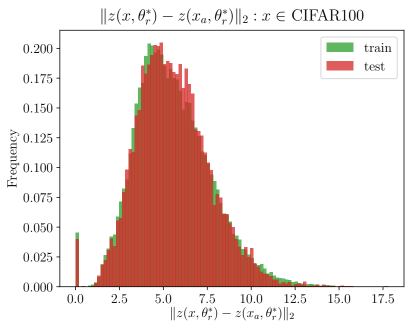
When freezing the feature extractor and fine-tuning on adversarial examples, we train the last fully connected layer's weights for 50 epochs using batch-size=128. We start with an initial learning rate of 0.01 and drop the learning rate to 0.001 at epoch 30. In the case of fine-tuning on adversarial examples, we generate the adversarial examples using a 3 step PGD attack with step-size 3 and a perturbation bound $\epsilon = 5$.

5.8.3 Free training experiments

In all of our free-training experiments where we train the u-ResNet-50, we train for 90 epochs using a batch-size of 128. The initial learning rate used is 0.1 and we drop it by a factor of 10 at epochs 30 and 60. We use a replay parameter $m = 4$ and perturbation bound $\epsilon = 5$.



(a) Histogram of $\|z(x, \theta_r^*), z(x_a, \theta_r^*)\|_2$ for CIFAR-10 dataset, given θ_r^* for CIFAR-100 dataset. The mean for both training and test examples is $\simeq 5.53$



(b) Histogram of $\|z(x, \theta_r^*), z(x_a, \theta_r^*)\|_2$ for CIFAR-100 dataset, given θ_r^* for CIFAR-100 dataset. The mean for both training and test examples is $\simeq 5.57$

Figure 5.7: Figures 5.7a and 5.7b both show that the values of $\|z(x, \theta_r^*), z(x_a, \theta_r^*)\|_2$ are high most of the time, consequently, LwF is better done without data augmentation.

5.9 The distance between feature representations of natural images and augmented images

To speed up the LwF experiments, we did not use data augmentation during training. Instead of computing the robust feature representations on the fly, before starting training on the new target task, we passed the entire training data of the target task through the robust network and stored the feature representation vector. If we were doing data augmentation, we would have to pass the entire augmented training data through the network, which would be slow and memory intensive. Alternatively, we could use the robust feature representation of the non-augmented images instead. The latter would have been feasible if the distance between the robust feature representations of the non-augmented and augmented images were very small. However, as shown in fig 5.7, this quantity is not often negligible.

Table 5.6: Distilling robust features using LwF for the split CIFAR-100 task. For reference, we have included the results from transfer learning by freezing the features at the bottom of the table.

Source $\rightarrow$ Target Dataset	Source Model	λ_d	val.	PGD-20
CIFAR-100+ (1/2) $\rightarrow$ CIFAR-100 (other 1/2)	robust	0.001	73.30%	1.92%
		0.005	66.96%	10.52%
		0.01	**63.32%**	**15.68%**
		0.1	55.14%	17.26%
CIFAR-100+ (1/2) $\rightarrow$ CIFAR-100+ (other 1/2)	natural	NA	71.44%	0.00%
	robust	NA	58.48%	15.86%

5.10 LwF-based robust transfer learning for similar source and target datasets

In Table 5.6 we conduct LwF experiments on the split CIFAR-100 task which is more suited for transfer learning due to the similarities between the source and target datasets. In these situations, the LwF regularizer on the feature representations still works and can improve generalization without becoming vulnerable to adversarial examples. If we take the average performance of the robust classifiers on the split tasks (average of robust half CIFAR-100 and the LwF setting model for $\lambda_d = 0.01$) we get $(63.32 + 64.96)/2 = 64.14\%$ average validation accuracy and 20.42% average robustness which is comparable with the case that we had adversarially trained the entire CIFAR-100 dataset (Table 5.1).

Table 5.7: Decreasing generalization gap by transferring with LwF. For reference, last row shows results from adversarial training CIFAR-10. The '+' sign refers to using augmentation.

Source $\rightarrow$ Target Dataset	Source Model	λ_d	val.	PGD-20
CIFAR-10+ $\rightarrow$ CIFAR-10	robust	1e-5	88.16%	45.31%
		5e-5	**88.08%**	**46.24%**
		1e-4	87.81%	46.54%
		5e-4	87.44%	46.36%
		0.001	87.31%	46.27%
		0.01	87.27%	46.09%
		0.1	87.49%	46.11%
CIFAR-10+	robust	NA	87.25%	45.84%

5.11 Improving generalization of the CIFAR-10 adversarially trained model

Similar to the case of improving the generalization for CIFAR-100, we use our LwF-based loss function to transfer from the robust CIFAR-10 domain to the natural CIFAR-10 domain. We summarize the results in Table 5.7.

Chapter 6: Conclusion

In this work, we discussed adversarial attacks in real-world settings. We discussed the pervasiveness of machine learning models in our everyday lives and the financial and legal implications of a malfunction in those systems. We explored some examples of such business-critical systems and their reliance on machine learning models. We also explained why malicious actors have financial incentives to attack these systems. We presented realistic attack models against machine learning models in real-world industrial systems. Finally, we proposed a novel approach for building robust models given realistic constraints.

We presented an example of a critical real-world system that relies on machine learning models: a voice assistant error resolver. Many times voice assistant systems misinterpret or misunderstand the intentions of a user. Since voice assistants are widely used by billions of users, ensuring and certifying the accuracy of these systems is crucial. However, the massive volume of data generated by these systems makes it impossible to manually detect such errors and fix them. To address that issue, we explained how a collaborative filtering model tackles voice assistant errors, and we showed the effectiveness of this approach against a large-scale, state-of-the-art voice assistant.

As a proof of concept, we demonstrated a realistic attack model against real-world recommendation systems. In most online stores, a recommendation system is responsible for deciding which items are presented to a user. Therefore, the sales revenue of an item is determined by these systems, which creates an incentive for malicious actors to try to fool the system to show their item to more users. To demonstrate a realistic attack by a malicious actor, we proposed a novel attack model that targets cold-start recommendations in a content-based model. In this approach, the bad actor can successfully fool the system without being detected by a human. We also addressed how such attacks can be avoided by leveraging adversarial training.

To further demonstrate the vulnerability of the real-world systems to adversarial attacks, we explored an attack against copyright detection systems. Given that copyright detection systems ensure that billions of dollars in revenue are distributed to the true copyright owners, there is a real incentive for malicious actors to attack these systems. We proposed a novel approach for crafting adversarial examples that can evade detection by real-world copyright detection systems. By using a gradient-based attack against a hand-crafted fingerprinting model that we proposed, we created adversarial music that is easily recognizable to a human while evading detection by a machine. We showed the effectiveness of adversarial examples crafted by our proposed approach against black-box models. Our examples successfully evaded detection by proprietary industrial systems.

Finally, we discussed how transfer learning can be used to train adversarially robust models. Given the data-hungry and computationally expensive nature of training robust models, we need a solution for real-world scenarios where we have access to little training data or we have a limited amount of computational resources. Transfer learning, in which a model is pre-trained on one dataset and is then re-trained on another dataset, is a widely used method to train complex models on small datasets. In this work, we showed how transfer learning can also be used to train robust models. We identified the feature extractors of adversarially trained models as a source of robustness, and use this observation to transfer robustness to new problems domains without adversarial training. The success of this method suggests that a robust feature extractor is effectively a filter that sifts out relevant components of an image that are needed to assign class labels.

Bibliography

[1] Tencent. Experimental security research of tesla autopilot. Technical report, Tencent Keen Security Lab, 2019.

[2] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *arXiv preprint arXiv:1611.01236*, 2016.

[3] Anish Athalye, Logan Engstrom, Andrew Ilyas, and Kevin Kwok. Synthesizing robust adversarial examples. *arXiv preprint arXiv:1707.07397*, 2017.

[4] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Chaowei Xiao, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Robust physical-world attacks on deep learning models. *arXiv preprint arXiv:1707.08945*, 2017.

[5] Hiromu Yakura and Jun Sakuma. Robust audio adversarial example for a physical attack. *arXiv preprint arXiv:1810.11793*, 2018.

[6] Yao Qin, Nicholas Carlini, Ian Goodfellow, Garrison Cottrell, and Colin Raffel. Imperceptible, robust, and targeted adversarial examples for automatic speech recognition. *arXiv preprint arXiv:1903.10346*, 2019.

[7] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. In *Joint European conference on machine learning and knowledge discovery in databases*, pages 387–402. Springer, 2013.

[8] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[9] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *arXiv preprint arXiv:1706.06083*, 2017.

[10] Ali Shafahi, Mahyar Najibi, Zheng Xu, John Dickerson, Larry S Davis, and Tom Goldstein. Universal adversarial training. *arXiv preprint arXiv:1811.11304*, 2018.

[11] Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Aleksander Madry. Adversarially robust generalization requires more data. In *Advances in Neural Information Processing Systems*, pages 5014–5026, 2018.

[12] Micah Goldblum, Avi Schwarzschild, Naftali Cohen, Tucker Balch, Ankit B Patel, and Tom Goldstein. Adversarial attacks on machine learning systems for high-frequency trading. *arXiv preprint arXiv:2002.09565*, 2020.

[13] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 577–585. Curran Associates, Inc., 2015. URL `http://papers.nips.cc/paper/5847-attention-based-models-for-speech-recognition.pdf`.

[14] Alex Graves. Sequence transduction with recurrent neural networks, 2012.

[15] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, May 2013. doi: 10.1109/icassp.2013.6638947. URL `http://dx.doi.org/10.1109/ICASSP.2013.6638947`.

[16] Alex Graves and Navdeep Jaitly. Towards end-to-end speech recognition with recurrent neural networks. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1764–II–1772. JMLR.org, 2014. URL `http://dl.acm.org/citation.cfm?id=3044805.3045089`.

[17] Jinfeng Rao, Ferhan Ture, and Jimmy Lin. Multi-task learning with neural networks for voice query understanding on an entertainment platform. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery &#38; Data Mining*, KDD '18, pages 636–645, New York, NY, USA, 2018. ACM. ISBN 978-1-4503-5552-0. doi: 10.1145/3219819.3219870. URL `http://doi.acm.org/10.1145/3219819.3219870`.

[18] Jinfeng Rao, Ferhan Ture, Hua He, Oliver Jojic, and Jimmy Lin. Talking to your tv. *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management - CIKM '17*, 2017. doi: 10.1145/3132847.3132893. URL `http://dx.doi.org/10.1145/3132847.3132893`.

[19] Ciprian Chelba and Johan Schalkwyk. *Empirical Exploration of Language Modeling for the google.com Query Stream as Applied to Mobile Voice Search*, pages 197–229. New York, 2013. URL `http://www.springer.com/engineering/signals/book/978-1-4614-6017-6`.

[20] Ido Guy. Searching by talking: Analysis of voice queries on mobile web search. In *Proceedings of the 39th International ACM SIGIR Conference on Research and*

Development in Information Retrieval, SIGIR '16, pages 35–44, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4069-4. doi: 10.1145/2911451.2911525. URL `http://doi.acm.org/10.1145/2911451.2911525`.

[21] Ahmed Hassan Awadallah, Ranjitha Gurunath Kulkarni, Umut Ozertem, and Rosie Jones. Characterizing and predicting voice query reformulation. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 543–552, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3794-6. doi: 10.1145/2806416.2806491. URL `http://doi.acm.org/10.1145/2806416.2806491`.

[22] Milad Shokouhi, Umut Ozertem, and Nick Craswell. Did you say u2 or youtube?: Inferring implicit transcripts from voice search logs. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 1215–1224, Republic and Canton of Geneva, Switzerland, 2016. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4143-1. doi: 10.1145/2872427.2882994. URL `https://doi.org/10.1145/2872427.2882994`.

[23] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pages 285–295. ACM, 2001.

[24] Jinfeng Rao, Ferhan Türe, and Jimmy Lin. What do viewers say to their tvs?: An analysis of voice queries to entertainment systems. In *SIGIR*, pages 1213–1216, 2018.

[25] Mihajlo Grbovic, Vladan Radosavljevic, Nemanja Djuric, Narayan Bhamidipati, Jaikit Savla, Varun Bhagwan, and Doug Sharp. E-commerce in your inbox: Product recommendations at scale. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '15, pages 1809–1818, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3664-2. doi: 10.1145/2783258.2788627. URL `http://doi.acm.org/10.1145/2783258.2788627`.

[26] O. Barkan and N. Koenigstein. Item2vec: Neural item embedding for collaborative filtering. In *2016 IEEE 26th International Workshop on Machine Learning for Signal Processing (MLSP)*, pages 1–6, Sept 2016. doi: 10.1109/MLSP.2016.7738886.

[27] Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, New York, NY, USA, 2016.

[28] David C. Liu, Stephanie Rogers, Raymond Shiau, Dmitry Kislyuk, Kevin C. Ma, Zhigang Zhong, Jenny Liu, and Yushi Jing. Related pins at pinterest: The evolution of a real-world recommender system. In *Proceedings of the 26th International Conference on World Wide Web Companion*, WWW '17 Companion, pages 583–592, Republic and Canton of Geneva, Switzerland, 2017. International World Wide

Web Conferences Steering Committee. ISBN 978-1-4503-4914-7. doi: 10.1145/3041021.3054202. URL `https://doi.org/10.1145/3041021.3054202`.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[30] Parsa Saadatpanah, Ali Shafahi, and Tom Goldstein. Adversarial attacks on copyright detection systems. In *International Conference on Machine Learning*, pages 8307–8315. PMLR, 2020.

[31] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. Wide & deep learning for recommender systems. In *Proceedings of the 1st workshop on deep learning for recommender systems*, pages 7–10, 2016.

[32] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th international conference on world wide web*, pages 173–182, 2017.

[33] Konstantina Christakopoulou and Arindam Banerjee. Adversarial attacks on an oblivious recommender. In *Proceedings of the 13th ACM Conference on Recommender Systems*, pages 322–330, 2019.

[34] Robin Van Meteren and Maarten Van Someren. Using content-based filtering for recommendation. In *Proceedings of the Machine Learning in the New Information Age: MLnet/ECML2000 Workshop*, volume 30, pages 47–56, 2000.

[35] Michael J Pazzani and Daniel Billsus. Content-based recommendation systems. In *The adaptive web*, pages 325–341. Springer, 2007.

[36] Michal Kompan and Mária Bieliková. Content-based news recommendation. In *International conference on electronic commerce and web technologies*, pages 61–72. Springer, 2010.

[37] Zeno Gantner, Lucas Drumond, Christoph Freudenthaler, Steffen Rendle, and Lars Schmidt-Thieme. Learning attribute-to-feature mappings for cold-start recommendations. In *2010 IEEE International Conference on Data Mining*, pages 176–185. IEEE, 2010.

[38] Jingjing Li, Mengmeng Jing, Ke Lu, Lei Zhu, Yang Yang, and Zi Huang. From zero-shot learning to cold-start recommendation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 4189–4196, 2019.

[39] Suvash Sedhain, Aditya Krishna Menon, Scott Sanner, Lexing Xie, and Darius Braziunas. Low-rank linear cold-start recommendation from social data. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[40] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. A meta-learning perspective on cold-start recommendations for items. In *Advances in neural information processing systems*, pages 6904–6914, 2017.

[41] Maksims Volkovs, Guangwei Yu, and Tomi Poutanen. Dropoutnet: Addressing cold start in recommender systems. In *Advances in Neural Information Processing Systems*, pages 4957–4966, 2017.

[42] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80, 2003.

[43] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, pages 1528–1540, 2016.

[44] James Bennett, Stan Lanning, and Netflix Netflix. The netflix prize. In *In KDD Cup and Workshop in conjunction with KDD*, 2007.

[45] Hao Wang, Naiyan Wang, and Dit-Yan Yeung. Collaborative deep learning for recommender systems. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1235–1244, 2015.

[46] Ke Zhou and Hongyuan Zha. Learning binary codes for collaborative filtering. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 498–506, 2012.

[47] Cedric Manara. Protecting what we love about the internet: our efforts to stop online piracy, 2018. `https://www.blog.google/outreach-initiatives/public-policy/protecting-what-we-love-about-internet-our-efforts-stop-online-piracy/` [Accessed: 05/21/2019].

[48] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283, 2016.

[49] AudioTag. Audiotag – free music recognition robot, 2009. URL `https://audiotag.info/`.

[50] Google. How content id works – youtube help, 2019. URL `https://support.google.com/youtube/answer/2797370?hl=en`.

[51] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *International Conference on Learning Representation*, 2015.

[52] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2574–2582, 2016.

[53] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Universal adversarial perturbations. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1765–1773, 2017.

[54] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Yuyin Zhou, Lingxi Xie, and Alan Yuille. Adversarial examples for semantic segmentation and object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1369–1378, 2017.

[55] Volker Fischer, Mummadi Chaithanya Kumar, Jan Hendrik Metzen, and Thomas Brox. Adversarial examples for semantic image segmentation. *arXiv preprint arXiv:1703.01101*, 2017.

[56] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *2018 IEEE Security and Privacy Workshops (SPW)*, pages 1–7. IEEE, 2018.

[57] Moustafa Alzantot, Bharathan Balaji, and Mani Srivastava. Did you hear that? adversarial examples against automatic speech recognition. *arXiv preprint arXiv:1801.00554*, 2018.

[58] Rohan Taori, Amog Kamsetty, Brenton Chu, and Nikita Vemuri. Targeted adversarial examples for black box audio systems. *arXiv preprint arXiv:1805.07820*, 2018.

[59] Jiajun Lu, Hussein Sibai, Evan Fabry, and David Forsyth. No need to worry about adversarial examples in object detection in autonomous vehicles. *arXiv preprint arXiv:1707.03501*, 2017.

[60] Jiajun Lu, Hussein Sibai, Evan Fabry, and David Forsyth. Standard detectors aren't (currently) fooled by physical adversarial stop signs. *arXiv preprint arXiv:1710.03337*, 2017.

[61] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[62] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *arXiv preprint arXiv:1904.12843*, 2019.

[63] Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. Certified adversarial robustness via randomized smoothing. *arXiv preprint arXiv:1902.02918*, 2019.

[64] Claudia Saviaga and Carlos Toxtli. Deepiracy: Video piracy detection system by using longest common subsequence and deep learning, 2018. `https://www.blog.google/outreach-initiatives/public-policy/protecting-what-we-love-about-internet-our-efforts-stop-online-piracy/` [Accessed: 05/21/2019].

[65] Derui Wang, Chaoran Li, Sheng Wen, Surya Nepal, and Yang Xiang. Daedalus: Breaking non-maximum suppression in object detection via adversarial examples. *arXiv preprint arXiv:1902.02067*, 2019.

[66] Y. Li, D. Wang, and L. Tang. Robust and secure image fingerprinting learned by neural network. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2019. ISSN 1051-8215. doi: 10.1109/TCSVT.2019.2890966.

[67] J. Yasaswi, S. Purini, and C. V. Jawahar. Plagiarism detection in programming assignments using deep features. In *2017 4th IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 652–657, Nov 2017. doi: 10.1109/ACPR.2017.146.

[68] Avery Wang et al. An industrial strength audio search algorithm. In *Ismir*, volume 2003, pages 7–13. Washington, DC, 2003.

[69] Tom Goldstein, Christoph Studer, and Richard Baraniuk. A field guide to forward-backward splitting with a fasta implementation. *arXiv preprint arXiv:1411.3406*, 2014.

[70] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[71] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[72] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[73] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 39–57. IEEE, 2017.

[74] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. *arXiv preprint arXiv:1802.00420*, 2018.

[75] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian Goodfellow, Dan Boneh, and Patrick McDaniel. Ensemble adversarial training: Attacks and defenses. *arXiv preprint arXiv:1705.07204*, 2017.

[76] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[77] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[78] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. Robustness may be at odds with accuracy. *stat*, 1050:11, 2018.

[79] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P Xing, Laurent El Ghaoui, and Michael I Jordan. Theoretically principled trade-off between robustness and accuracy. *arXiv preprint arXiv:1901.08573*, 2019.

[80] Dong Su, Huan Zhang, Hongge Chen, Jinfeng Yi, Pin-Yu Chen, and Yupeng Gao. Is robustness the cost of accuracy?–a comprehensive study on the robustness of 18 deep image classification models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 631–648, 2018.

[81] Ali Shafahi, W Ronny Huang, Christoph Studer, Soheil Feizi, and Tom Goldstein. Are adversarial examples inevitable? *ICLR*, 2019.

[82] Dan Hendrycks, Kimin Lee, and Mantas Mazeika. Using pre-training can improve model robustness and uncertainty. *arXiv preprint arXiv:1901.09960*, 2019.

[83] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[84] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[85] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[86] Zhizhong Li and Derek Hoiem. Learning without forgetting. *IEEE transactions on pattern analysis and machine intelligence*, 40(12):2935–2947, 2018.

[87] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.