Preparing Network Intrusion Detection Deep Learning Models with Minimal Data Using Adversarial Domain Adaptation

Ankush Singla
Department of Computer Science
Purdue University
asingla@purdue.edu

Elisa Bertino
Department of Computer Science
Purdue University
bertino@purdue.edu

Dinesh Verma
Thomas J. Watson Research Center
IBM, Yorktown Heights
New York, USA
dverma@us.ibm.com

ABSTRACT

Recent work has shown that deep learning (DL) techniques are highly effective for assisting network intrusion detection systems (NIDS) in identifying malicious attacks on networks. Training DL classification models, however, requires vast amounts of labeled data which is often expensive and time-consuming to collect. Also, DL models trained using data from one type of network may not be able to identify attacks on other types of network or identify new families of attacks discovered over time. In this paper, we propose and evaluate the use of adversarial domain adaptation to address the problem of scarcity of labeled training data in a dataset by transferring knowledge gained from an existing network intrusion detection (NID) dataset. Our approach works for scenarios where the source and target datasets have same or different feature spaces. We demonstrate that our proposed approach can create highly accurate DL classification models even when the number of labeled samples in the target dataset is significantly small.

CCS CONCEPTS

• **Security and privacy → Intrusion detection systems**; • **Computing methodologies → Transfer learning**; **Neural networks**.

KEYWORDS

Deep Learning, Intrusion Detection, Transfer Learning, Neural Networks

ACM Reference Format:
Ankush Singla, Elisa Bertino, and Dinesh Verma. 2020. Preparing Network Intrusion Detection Deep Learning Models with Minimal Data Using Adversarial Domain Adaptation. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20), October 5–9, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3320269.3384718

1 INTRODUCTION

Network intrusion detection systems (NIDS) detect malicious activity in a network (denial-of-service, probing, unauthorized access etc.) by monitoring and analyzing the traffic flowing through the network. Traditional NIDS suffer from several drawbacks, including: inability to detect new or unseen attacks, as in the case of signature-based NIDS; time-consuming training phase and huge numbers of false positives, as in the case of anomaly-based NIDS. Deep learning (DL) based techniques have thus been proposed to aid NIDS in addressing those drawbacks [6, 18, 36]. DL classification models learn from labeled training data containing samples of malicious and benign network traffic and apply this knowledge to categorize new unseen data at runtime.

Training DL models, however, requires large amounts of labeled data. Collecting and labeling data for different newer attack families is often very time consuming and costly. In addition to this, labeled data from one type of network may not be suited for training DL models for another type of network. For example, a dataset used to train a DL model for a traditional network may not work well for mobile or IoT networks. Furthermore, very often the attack distribution changes with time as the types of network devices, protocols and attack vectors evolve, thus requiring new labeled data to keep the DL models up to date.

In this paper, we propose the use of *Domain Adaptation* (DA) to reduce the amount of labeled data required for training DL network intrusion detection (NID) models. DA is a transfer learning technique that allows one to transfer knowledge from a *source domain* with adequate training data, to a different but similar *target domain* with minimal or no new training data. For example, a DL object detection model prepared for identifying traffic signs by training on a labeled dataset containing only images from USA might not perform well in another country because of differences in background, language and traffic rules. However, the source DL model can be re-purposed to operate in the target country by using DA and only a small labeled set of traffic sign images from that country. We already have a wide variety of public NID datasets [42] that contain labeled data for identifying various different attack categories. Leveraging these existing NID datasets, DA can enable us to prepare up-to date and highly accurate DL models for NID that can identify the latest malicious attacks being discovered in the wild as well as be able to work in specific types of network, like IoT and mobile networks, with minimal new training data. In our work, we use a specific form of DA, referred to as *adversarial DA*, that leverages generative adversarial networks (GANs) [16] for creating a domain-invariant mapping of the source and target datasets.

In this paper, we consider two broad scenarios to evaluate the efficacy of adversarial DA in training accurate NID models based on whether the source and the target datasets have a similar or different feature space. ❶ We have a source NID dataset with a large amount of labeled data samples and a target dataset with a few labeled data

ASIA CCS '20, October 5–9, 2020, Taipei, Taiwan

ACM ISBN 978-1-4503-6750-9/20/10…$15.00
https://doi.org/10.1145/3320269.3384718

samples with the same *same feature space*. This is applicable to cases in which the target dataset is created by collecting packet data from networks using the same protocols as the source dataset, maybe at a later period in time. The target dataset may contain new attack types that were not present in the source dataset. In this case, we want to be able to accurately detect both the new attack types as well as the old attack types. ❷ In other cases, the source NID dataset might have a *different feature space* than the target NID dataset. This scenario applies to cases in which the network protocols and the types of devices in the networks are different for the source and target datasets, thus changing the feature space; for example IoT networks might use different protocols than traditional networks. In addition, the tools used to capture and process network traffic might evolve allowing the capture of much more meaningful features.

In the paper, we present an adversarial DA methodology for training highly accurate DL models for both these scenarios when the target NIDS has minimal or no training data. We first apply data pre-processing techniques to prepare the source and target datasets for adversarial DA and apply a dimensionality reduction technique called Principal Component Analysis (PCA) to reduce the feature dimensions of our datasets. In the case of source and target datasets with different numbers of feature dimensions, PCA will serve the additional purpose of generating intermediate data representations that have an equal number of dimensions. We then use adversarial DA to map the source and target datasets into a domain-invariant representation using a custom GAN architecture, which results in a DL classification model that can categorize the benign and attack samples in both the source and the target datasets accurately.

Our paper has the following technical contributions:

(1) We introduce an adversarial DA methodology to train highly accurate DL models with minimal new labeled data for a NID in two specific scenarios:
   - Source and target datasets with similar feature spaces.
   - Source and target datasets with different feature spaces.
(2) We implement and test the described techniques on two popular publicly available NID datasets.
(3) We evaluate the efficacy of the suggested DA techniques with respect to the amount of data available in the target dataset.

The rest of the paper is organized as follows. We introduce terminology and basic concepts relevant to the paper in Section 2. We then discuss related work in Section 3. We provide the detailed description of our approach and techniques in Section 4. We then describe the datasets selected for our evaluation, experimental setup and other details in Section 5. We discuss the results of our experiments in the Section 6. Finally, we outline conclusions and future work in Section 7.

2 BACKGROUND

In this section we introduce the concepts and terminology relevant to our methodology for training DL models for NID.

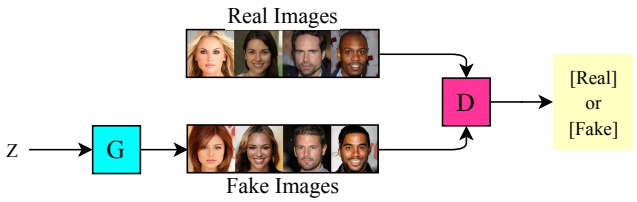

Figure 1: Architecture of a generative adversarial network (GAN). The generator G learns to produce fake images from a random noise vector z to fool the discriminator D, whose goal is to differentiate between real and fake images.

2.1 Network intrusion detection systems (NIDS)

Network intrusion detection systems monitor a network or a system for malicious attacks or policy violations. NIDS are broadly classified into *misuse-based* and *anomaly-based*. Misuse-based NIDS look for some specific patterns or signatures of previously identified attacks in the network traffic, system calls, etc. Misuse-based NIDS however, require upto date databases containing rules and signatures for all the attacks, making them ineffective against 0-day attacks. Anomaly-based techniques learn the normal network behavior during a training phase and then identify deviations from the normal behavior in the network behavior at run-time. Unlike misuse-based NIDS, they can detect newer attacks; however, they can have high false positive rates (FPR) as they can flag any unseen yet benign network behavior or system use as a potential malicious attack. They have a time-consuming training phase and need to be trained separately for every network deployment.

2.2 Deep Learning (DL)

Neural networks (NN) are systems inspired by the human brain and how the network of neurons processes information and performs computations [50]. Basic feed-forward NNs contain artificial neurons containing a mathematical transformation function organised in different layers. The layers between the input and the output layers are known as hidden layers. Each hidden layer receives input from the previous layers, uses its transformation function on them and passes its output to the next layers. Each neuron learns and optimizes the weights and biases for its transformation function during the training phase to minimize some cost function measuring the distance between the values generated by the NN and the actual values. A *deep neural network (DNN)* is an NN with more than one hidden layer between the input and output layers. *Deep learning (DL)* is a sub-field of machine learning (ML), based on the use of DNNs.

2.3 Generative adversarial networks (GAN)

An *adversarial nets* framework [16] consists of two NN models: a generative model called *generator* (G) and a discriminative model called *discriminator* (D). These NN models are pitted against each other in an adversarial game over a certain real data distribution. The generator's goal is to generate a synthetic data sample that

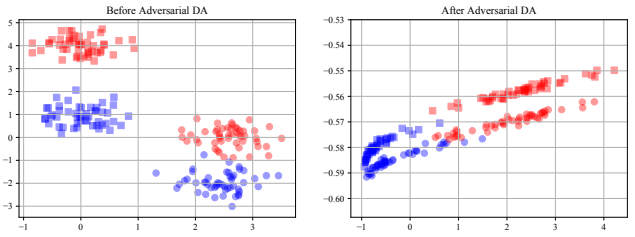

Figure 2: This figure illustrates how adversarial DA algorithms map two separate datasets into a common domain-invariant mapping. The two datasets are represented by squares and circles, and each has two classes represented by colors blue and red.

looks like it belongs to the real data distribution. The discriminator's goal is to check any data record and distinguish whether it is a real data sample or generated by G. The generator and the discriminator learn their tasks simultaneously and get better at their respective goals. The goal of this adversarial game is for G to produce synthetic data that D cannot distinguish from the original data. Fig. 1 gives an example of a GAN that takes fake images that look like they belong to the real data distribution. The generator G takes a random vector z as an input and generates a set of fake images. The discriminator D takes the real and fake generated images as the input and tries to classify them into real and fake categories. The training of GANs is modeled as a mini-max game where G and D are trained simultaneously using back-propagation, and G tries to minimize the value function $V(G, D)$ provided below, while D tries to maximize it:

$$\min_G \max_D V(G, D) = E_x[log\ D(x)] + E_z[log(1 - D(G(z))]$$

where: E_x is the expected value over all real data instances, E_z is the expected value over all the fake generated data instances, $D(x)$ is the probability of D predicting a real instance as real, and $D(G(z))$ is the probability of D predicting a fake generated instance as real.

GANs have been shown to be effective at generating synthetic images indistinguishable from real images by training on image databases like MNIST [28], CelebA [23], and CIFAR-10 [26]. GANs have also been used in language and speech processing [33, 38, 53], domain adaptation [4, 19, 45, 49], and data augmentation [11, 12, 31].

2.4 Domain adaptation (DA)

The concept of transfer learning (TL) allows ML algorithms to use the knowledge gained by learning one or more *source* tasks for learning a *target* task. TL is especially useful when we do not have enough training data in a particular domain, but there is a lot of training data available in another different but similar domain. TL has been shown to significantly improve ML algorithm accuracy in applications like image classification [35, 41], text classification [8, 10], and speech recognition [9, 20].

In any TL scenario, the source and target may either differ in *domain* (the feature space and the feature distribution of the datasets) or *task* (the label space and the objective predictive function learnt from the training data). Pan et al. classify TL into three broad categories based on the difference between the source and target

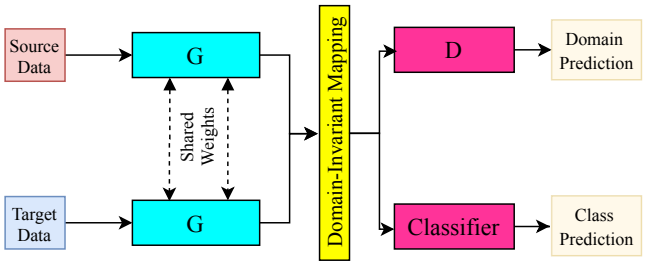

Figure 3: Architecture of the generic framework for adversarial DA using GANs. The generator(s) G learns (learn) to produce a common domain-invariant mapping of the source and target datasets. This mapping is used as an input for the discriminator D which tries to predict whether the sample belongs to the source or the target domain. The mapping is also used as an input to a classifier which predicts which class the sample belongs to. The final goal is to minimize both the domain prediction and the class prediction loss.

domain, task and availability of labeled data [37]. ❶ *Inductive*: the target domain can be similar or different from the source domain but the target task is always different from the source task, and some labeled data is available in the target domain; ❷ *Transductive*: the target domain is related but different from the source domain but the source and target tasks are the same; ❸ *Unsupervised*: the target and source domains and tasks are different and no labeled data is available in either source and target domains [37].

Domain adaptation (DA) is a special case of transductive TL i.e. the source and target tasks are the same, and the domains are related but different. DA can be further categorized into *homogeneous* and *heterogeneous*, based on whether the source and target domains have the same or different feature spaces, respectively [51]. Based on the amount of labeled data present in the target domain, each of these categories can be further classified into *supervised DA* (small amount of labeled data available in the target domain), *semi-supervised DA* (a small amount of labeled data and a lot of unlabeled data are available in the target domain), *unsupervised DA* (no labeled data but an adequate amount of unlabeled data is available in the target domain).

Most DA methods try to align the source and target data distributions to learn a domain-invariant mapping of the datasets by one of following three methods. ❶ by minimizing some distance metric that calculates domain shift among the source and target data distributions [17, 44]. ❷ By reconstructing the target and/or source data into a domain-invariant mapping [5, 14, 15]. ❸ By using GANs to map the source and target data distributions into a common subspace by using domain confusion [21, 29, 48].

Adversarial DA is a class of algorithms that leverages GANs to learn a common domain-invariant mapping between the source and target datasets (see Fig. 2), using an adversarial learning framework. Most adversarial DA approaches employ some version of a generic GAN framework (shown in Fig. 3) differing by: whether the underlying models are generative or discriminative, which loss functions are used, and whether weights are shared between G and D [48]. The CoGAN framework uses two GANs, in which G and D

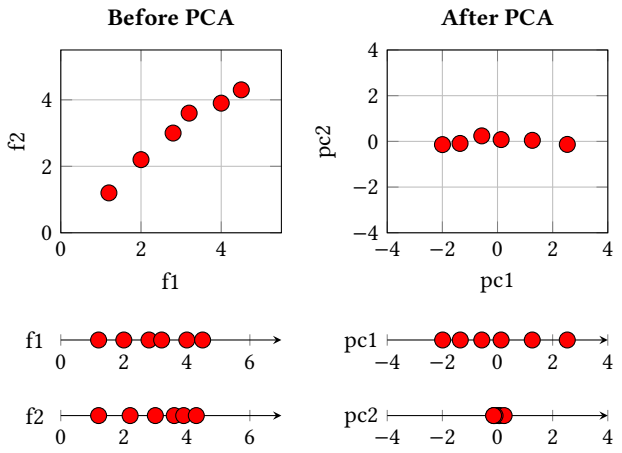

Figure 4: Principal component analysis (PCA) transformation. The first graph plots two features, f1 and f2, and the second graph plots the two principal components pc1 and pc2 after applying PCA transformation. The lines below the graphs show the features and principal components plotted individually to demonstrate variability. Source: [40]

from the respective GANs share the weights in the first few layers and learn to generate synthetic images in the source and target distributions simultaneously from input noise [29]. The adversarial discriminative DA (ADDA) approach completely uncouples the source and target G weights while keeping a single D [48], whereas domain-adversarial training of neural networks (DANN) approach uses a single feature extractor and two classifiers - one for label prediction and one for domain prediction [13]. The goal is to minimize the combined loss, i.e. the domain prediction loss and the class prediction loss, while learning a domain-invariant feature extractor for both the source and the target domains. All these approaches use convolutional neural networks (CNNs) as the underlying models for G which might not be suitable in case of non-image data. In addition to this, most of the DA approaches mentioned above are unsupervised (no labeled data in target domain) and thus, do not achieve a high enough accuracy on the target data. In this paper, we assume that we only have a small amount of labeled samples in the target domain. We instantiate a hybrid version of the DANN and ADDA architectures to cater to our specific scenario.

2.5 Principal component analysis (PCA)

When trying to predict a certain dependent variable using ML, there might be a plethora of independent variables (features) to consider. For example, when trying to predict the likelihood of a certain disease in a person, one might collect features like age, height, weight, gender, ethnicity, allergy information, medical history, hereditary predisposition to the disease, genetic information, etc. Training on a large number of inter-related features is likely to increase the complexity of the ML model and cause over-fitting, resulting in degradation of performance on unseen data. Various *dimensionality reduction* techniques have been proposed to reduce the number of features considered for training an ML model. These techniques mainly belong to one of two categories: *feature selection* and *feature extraction*. Feature selection refers to reducing the feature space by removing the features that do not have a significant correlation with the dependent variable and keep only the top k features on the basis of correlation to the output. This can be done manually or programmatically. However, feature selection results in loss of any information contained in the removed features. Feature extraction on the other hand allows one to convert a high-dimensional feature space into a low-dimensional representation without a significant loss of information.

PCA [39] is a feature extraction technique that performs a linear mapping from a high-dimensional feature space to a low-dimensional one. PCA takes a dataset consisting of n features and converts them into new linearly uncorrelated features, also known as *principal components*. One then selects the top k principal components (where k≤n) in descending order of variance to account for most of the variability in the input dataset in order to remove the least important principal components. These k selected principal components serve as the new features for the dataset. For brevity, we omit the mathematical explanation of PCA from this paper and refer the readers to the paper by Wold et al. [52].

Fig. 4 shows an example of the PCA transformation. Suppose we have two features f1 and f2 in a dataset. Both f1 and f2 represent almost equal variance in the dataset as seen from the individual plots of f1 and f2. PCA converts these into two principal components pc1 and pc2, where pc1 represents most of the variance in the dataset and pc2 represents very little variance. Thus, pc2 can be dropped allowing us to transform the dataset containing two features f1 and f2 into another dataset with just one feature pc1 and still account for most of the variance in the dataset. Similarly, we can use PCA to transform datasets with a much larger number of features and reduce the number of dimensions of the dataset while preserving the information represented by the dataset.

3 RELATED WORK

Previous approaches have been proposed that use NN models for both misuse-based NID [6, 18, 36] and for anomaly-based NID [24, 47]. However, these approaches assume the availability of enough labeled data for training their respective NN models and do not explore the problem of lack of training data.

Singla et al. evaluate the quality of DL models for NIDS trained using very low amounts of training data [43]. They propose to first train the DNN model on a source domain (similar to the target domain) with adequate training data and then use transfer learning to fine-tuning the model on the target data. Their technique, however, can only be applied to scenarios where the source and the target datasets have the same feature space.

Zhao et al. propose the CeHTL [55] and HeTL [54] transfer learning frameworks for training NID models with no labeled training data in the target domain, for both homogeneous and heterogeneous DA cases but they do not achieve a high accuracy on the target dataset. Both these approaches are unsupervised DA techniques (applicable only on unlabeled training data is the target domain) and do not look at supervised DA cases where a little amount of labeled training data is available.

Tzeng et al. [49] and Ganin et al. [13] propose the adversarial DA frameworks ADDA and DANN for unsupervised DA. However, both these approaches are primarily applicable to unsupervised DA cases. Moreover, these approaches are only evaluated on image classification tasks using convolutional neural networks (CNN) and not on domains with non-image data.

To the best of our knowledge, our approach is the first to use adversarial DA for training DL models for NID when training data is limited, both for the homogeneous and heterogeneous DA cases.

4 ADVERSARIAL DA METHODOLOGY

Our methodology consists of two steps. We first clean and transform the source and target datasets to be prepared for DA. We then use our adversarial DA framework to map the source and target datasets into a common latent subspace and consequently generate a NID classification model that can accurately categorize the benign and attack samples in the target dataset. In what follows we describe those steps in detail.

4.1 Data pre-processing

We first pre-processes both the source and target datasets to deal with missing entries from the datasets. We then transform the features depending on whether they are categorical or continuous and finally apply PCA for dimensionality reduction.

- *Handling missing values:* A real-world dataset might have a lot of values missing for several features, which can cause problems while training ML models, as a lot of ML libraries do not have provisions for handling missing data. There are three common techniques to deal with missing values. The first is remove all data records that have missing values. The second is to initialize the missing values by 0, but 0 might have some other meaning in the case of numerical or binary variables. For example, a binary variable like `is_logged_in` is represented by values 0 and 1 and setting the missing values as 0 might be misleading. The third is to set the missing values to the mean of all the values of that variable in the dataset; however, the application of this technique may also lead to misleading data. In our approach, we simply drop the dataset records that contain any missing values.
- *Transforming categorical features:* Categorical features are the features that have discrete values instead of continuous values. For example, `protocol_type` is a categorical feature since it can have only three possible values in our NID dataset i.e. TCP, UDP and ICMP. We need to convert these string values to integers as ML models cannot train on non-numeric data. However, simply converting these features to integer values can cause unexpected behavior from the ML models, since there is no ordinal relationship between them. To transform categorical features, we use one-hot encoding which converts categorical features to multiple binary features with values 0 or 1. Using one-hot encoding on our example, the feature `protocol_type` is replaced by three features: [`protocol_type_TCP, protocol_type_UDP` and `protocol_type_ICMP`]. The values of TCP, UDP and ICMP will be represented as [1, 0, 0], [0, 1, 0] and [0, 0, 1] respectively, in terms of the new features.
- *Processing continuous features:* Continuous features can assume an infinite number of values, for example: `duration` and `packet_count`. These features might have different scales for values like hours, metres, pounds etc. and can range over thousands of units. This can cause poor performance for the ML models and cause the weight values of the model variables to change drastically during training making the model unstable. Thus, we need to standardize the continuous variables to convert them to a common scale and map the input values in a certain small range. Standardization transforms the continuous features so that the resulting feature distribution has a mean of 0 and a standard deviation of 1. To standardize a feature, we first calculate the mean and standard deviation of all the numeric entries of the feature in the dataset. We then subtract the mean of the feature from each entry and divide this value by its standard deviation. $x' = \frac{x - \overline{x}}{\sigma}$ where, $\overline{x}$ is the mean and σ is the standard deviation of the feature.
- *PCA:* We apply PCA transformation on the source and target datasets to reduce their dimensions to some common value d, where $d \leq dim(source)$ and $d \leq dim(target)$. Here, $dim(source)$ and $dim(target)$ refer to the original dimensions of the source and target datasets respectively. PCA essentially converts the problem of heterogeneous DA to homogeneous DA by allowing our adversarial DA technique to be applied to the any DA problem regardless of the dimensions. PCA also converts the possibly correlated features of the datasets into completely uncorrelated features, which is a requirement for some ML algorithms. The common dimension d is chosen carefully so that the transformed dataset still explains most of the variance in the original dataset.

4.2 Adversarial domain adaptation

In this section, we describe the GAN architecture and loss functions we use for adversarial DA.

GAN architecture: Our DA GAN architecture consists of a generator and discriminator that are DNNs with the same layer configuration (see Fig 5). They both consist of 9 layers with 3 sets of fully connected layers, a batch normalization layer, and a reLU activation layer stacked on top of each other. The fully connected layers have 64, 32, 16 neurons in that order. The last layer of the generator serves as input to the discriminator as well as feeds into a soft-max layer to predict the class of the data sample. Similarly, the final layer of the discriminator feeds into a soft-max layer to predict the domain the sample belongs to.

GAN training: Our GAN implementation's main objective is to learn a NID classifier that can accurately predict whether the data samples in the target domain belong to the attack or benign class. It does this by trying to learn a mapping between source and target domains, so that they can be transformed into a common latent subspace. This allows us to overcome the lack of labeled data in the target domain by using the mapped data from the source domain for training. The goal of the generator in our GAN architecture is to take samples from the source and target data distributions and convert them into a domain-invariant representation to fool

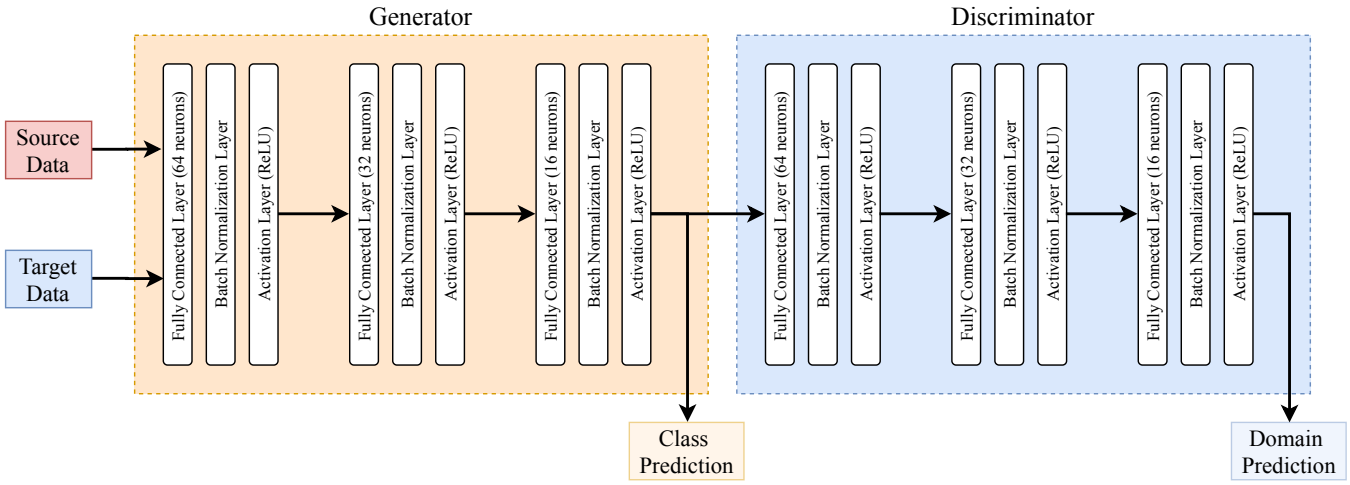

Figure 5: Our GAN architecture for adversarial DA. We train the GAN to minimize the class prediction and domain prediction loss. The adversarial DA process outputs the generator as a classifier for the target dataset.

Algorithm 1: Algorithm for adversarial domain adaptation using our GAN architecture.

input : Adam optimizer has learning rate l and decay rates β_1 and β_2
output: The generator G as a classifier
for *n training iterations* **do**
  Select d samples from the source dataset;
  Select d samples from the target dataset;
  Decrease the discriminator gradient using Adam optimiser using loss:
  $-\frac{1}{d}\sum_{i=1}^{d}[\log D(G(x_s^{(i)}))] + [\log(1 - D(G(x_t^{(i)})))]$
  Decrease the generator gradient using Adam optimiser using loss:
  $\frac{1}{2}[L_{class} + L_{domain}]$
  where, $L_{class} = -\frac{1}{2d}\sum_{i=1}^{2d} y^{(i)}[\log(G(x^{(i)})] + (1 - y^{(i)})[\log(1 - G(x^{(i)})]$
  and, $L_{domain} = -\frac{1}{d}\sum_{i=1}^{d}[\log(D(G(x_t^{(i)})))]$
end

the discriminator into misclassifying the generated representations. The discriminator's goal is to identify whether the representation provided by the generator belongs to the source or the target data distribution. The generator also has the additional objective of being able to distinguish between the attack and benign classes on the source and target data distributions. The generator and the discriminator are trained simultaneously to get better at their respective tasks.

The formal algorithm for our adversarial DA technique is provided in Algorithm 1. The algorithm iterates over the source and target dataset. In each iteration, the algorithm selects a batch of data samples from the source and the target data distributions equal to a predefined batch size d. The generator updates its weights to maximize the domain loss (i.e. error produced by the domain predictor) and minimize the classification loss (i.e. error produced by the class predictor) on this batch. The discriminator weights remain frozen during this step. Simultaneously, the discriminator updates its weights to minimize the domain loss on the current batch. The generator weights remain frozen during this step. After n iterations, the algorithm outputs the generator as a NID classification model.

Loss functions: We define the various loss functions for training our GAN architecture in this section. We first define two loss functions, domain loss L_{domain} and classification loss L_{class} in equations 1 and 2 below. The domain loss function measures the difference between real values and the predicted values of the domain generated by our discriminator. Here, domain prediction refers to the probability of the data sample belonging to the source or the target data distribution. Similarly, the classification loss function measures the difference between the real and the predicted values of the class predictions by our generator. Here, class prediction refers to the probability values of the data sample belonging to the attack or benign category.

The **domain loss** is calculated as:

$$L_{domain}(G, D) = -E_{x_s}[\log D(G(x_s))] - E_{x_t}[\log(1 - D(G(x_t))] \quad (1)$$

where E_{x_s} and E_{x_t} are the expected values of the source domain samples and the target domain samples respectively, $D(G(x_s))$ is

the probability of the discriminator predicting a source domain sample as belonging to source, and $D(G(x_t))$ is the probability of the discriminator predicting a target domain sample as belonging to source.

The **classification loss** is calculated as:

$$L_{class}(G) = -E_{x_{attack}}[\log(G(x))] - E_{x_{benign}}[\log(1 - G(x))] \quad (2)$$

where $E_{x_{attack}}$ and $E_{x_{benign}}$ are the expected values of attack and benign samples respectively over the combined source and target true data distributions, $G(x)$ is the probability of the generator classifying a sample as attack, and $1 - G(x)$ is the probability of the generator classifying a sample as benign.

The domain loss described in equation 1 can be split into two minimization objectives for the generator and discriminator if we invert the domain labels while training the generator (see equations 3 and 4) [16]. Our algorithms train the generator and discriminator simultaneously according to the following three minimization objectives:

$$\min_D L_{domain}(G) = -E_{x_s}[\log D(G(x_s))] - E_{x_t}[\log(1 - D(G(x_t))] \quad (3)$$

$$\min_G L_{domain}(D) = -E_{x_t}[\log(D(G(x_t))] \quad (4)$$

$$\min_G L_{class} \quad (5)$$

5 EXPERIMENT DETAILS

We now provide the details of the datasets used for our experiments, details of the experiments, the metrics collected, and the system setup.

5.1 Selecting the source and target datasets

There are several public datasets available for NIDS evaluation. *KDD-CUP99* [27] has been the most popular NIDS bench-marking dataset for many years and was created by MIT Lincoln Labs using a U.S. Air Force LAN simulation. However, it suffers from various discrepancies such as redundant records and different probability distributions for the training and testing datasets [30, 32]. *NSL-KDD* [1] was created by addressing the discrepancies of the KDD-CUP99 dataset.

The 10% NSL-KDD dataset contains 125,973 records in the training dataset and 22,544 records in the testing dataset. It contains 41 features categorized as: ❶ basic features of individual connections (e.g. duration, protocol type, service), ❷ content features suggested by domain knowledge (e.g. number of failed login attempts, number of shell prompts) and ❸ traffic features computed using a 2 seconds window (e.g. number of connections to the same host, percent of connections that have SYN errors). The NSL-KDD dataset categorizes records into five major attack categories: normal, denial of service (e.g. syn flood attack, neptune, teardrop), probing (e.g. port scanning, satan), remote-to-local (e.g. guessing password, warezclient, warezmaster), and user-to-root (e.g. buffer overflow, rootkit) (see Table 1).

The *UNSW-NB15* dataset [34] is a NIDS evaluation dataset containing more recent attack types and distributions than the NSL-KDD dataset. This dataset was created using the IXIA PerfectStorm security testing platform [22] that can generate simulated traffic

Category	Percentage	Number of records
Normal	53.45 %	67343
DoS	36.45 %	45927
Probing	9.25 %	11656
Remote-to-local	0.78 %	995
User-to-root	0.04 %	52

Table 1: Distribution of attack categories (NSL-KDD training dataset).

Category	Percentage	Number of records
Normal	31.94 %	56000
Generic	22.81 %	40000
Exploits	19.04 %	33393
Fuzzers	10.37 %	18184
DoS	6.99 %	12264
Reconnaissance	5.98 %	10491
Analysis	1.14 %	2000
Backdoor	1.0 %	1746
Shellcode	0.65 %	1133
Worms	0.07 %	130

Table 2: Distribution of attack categories (UNSW-NB15 training dataset).

for several attack scenarios. The partitioned UNSW-NB15 dataset contains 175,341 records in the training set and 82,332 records in the testing set. It contains 42 features categorized as ① flow features (e.g. protocol, source/destination port), ② basic features (e.g. state, duration, service), ③ content features (e.g. source/destination TCP sequence number), ④ time features (e.g. source/destination jitter, record start/end time), and ⑤ additional generated features (e.g. number of flows that have GET and POST methods in the HTTP service, number of connections that have the same destination address). The UNSW-NB15 dataset categorizes records into normal and 9 attack categories (see Table 2). The features of UNSW-NB15 and NSL-KDD are described in the appendix Table 4 and Table 5, respectively.

5.2 Description of experiments

We divide our experiments into two categories based on whether the feature spaces of the source and target datasets are similar or different.

- **Same feature space:** This category represents the cases where the source NID dataset has the *same feature space* as the target NID dataset. This is typical when the target dataset is created by collecting packet data from networks deploying the same protocols and using the same processing tools as the source dataset, maybe at a later period in time. An example of this scenario is where an organization creates a source dataset that has data samples belonging to 10 attack categories possible on its network and trains a classification model on it, but at a later time the organization discovers 2 new categories of attacks. The organization then collects a small amount of data from those 2 new attack categories

and wants to create a new NID classification model that can identify all 12 attack categories.
For the experiments in this category, we split the UNSW-NB15 dataset into two parts: a source dataset with samples from 8 attack categories except samples of the attack category *exploits* and a target dataset with samples of the attack category *exploits*. We use our adversarial DA technique to train a NN classification model using these source and target datasets and evaluate its performance in detecting attacks of the type *exploits* belonging to the target dataset, as well as the attacks from the 8 attack categories present in the source dataset. We do this for different target data sizes to show the impact of data size on the accuracy of the target model. We repeat the experiments for attack categories: *reconnaissance* and *shellcode* to demonstrate the generality of our approach. We performed these experiments for all 9 attack categories in the UNSW-NB15 dataset and observed similar trends.

- **Different feature space:** This category represents the cases where the source NID dataset has a *different feature space* than the target NID dataset. This scenario is more common when the source and the target datasets are either created for different network types using different network protocols or the tools used to capture and process the network traffic for monitoring have evolved allowing the capture of much more meaningful features, thus changing the feature space. The attack distributions might also completely change with time due to the older attacks becoming irrelevant due to being patched. Also attackers identifying new attack vectors might give rise to completely new attack families. An example of this scenario is an organization that already has a source dataset of data samples belonging to several attack categories collected for their internal employee WiFi network and train a NID classification model on this dataset. They now want to create a NID classification model that can identify attacks on their experimental IoT network. So they collect a small amount of samples of attacks for the IoT network and create a target dataset. This target dataset will have a different feature space than the source dataset because the protocols for IoT devices are different. We use our adversarial DA technique to train on the source and the target dataset to create a NID classification model that can identify attacks in the target IoT networks. For experiments in this category, we use NSL-KDD as the source dataset and UNSW-NB15 as the target dataset. We use our adversarial DA technique to train a NN classification model and evaluate its performance when different sizes of target datasets are used to demonstrate the impact of data size on the performance of the model.

5.3 Libraries and hyper-parameters

We use the open source DL library TensorFlow [2] with the Keras [7] wrapper to train our GAN and other classification models for comparison. We use a standard 75-25 % split for training-testing sets for our experiments. We use 60 epochs for training the base model with a batch size of 32. For our DA GAN, we train for 10000 iterations with a batch size of 32. We use the adaptive learning rate optimization algorithm Adam [25], with a starting learning rate of 0.001, β_1 = 0.9 and β_2 = 0.999 for training our NNs and the GAN. For describing the performance of the trained classification model, we report the accuracy and F-score metrics.

6 RESULTS AND ANALYSIS

We compare our adversarial DA approach (described in Section 4) with: ❶ the base case, that is, a DL model trained only on the target dataset; and ❷ a transfer learning approach that uses fine-tuning [43]. The results are divided into two categories based on whether the source and target datasets have the *same feature space* or *different feature space*. For all three approaches in our evaluation, we use the same classification model architecture. The model consists of a total of 9 layers with 3 sets of fully connected layers, a batch normalization layer, and a reLU activation layer stacked on top of each other. The fully connected layers have 64, 32, 16 neurons in that order. The output layer of the model is a layer with 2 neurons and a soft-max activation function to give the probability predictions of the data sample belonging to the attack or benign category.

6.1 Same feature space

This category represents the scenario where the source and target datasets have the same feature space.
Datasets: For this evaluation, we split UNSW-NB15 [34] into two parts. One dataset with the data samples from 8 attack categories except *exploits* (serves as the source dataset) and one with only samples of the attack category exploits (serves as the target dataset). Both the source and the target datasets contain samples of the category benign. We repeat similar experiments for other two attack categories: *reconnaissance* and *shellcode*. We model this as a binary classification problem, i.e. predicting whether the current record belongs to the attack or benign category, thus not predicting the specific attack category. In case of the attack category exploits, the source training dataset has 83,961 labeled samples. Similarly, in case of attack category reconnaissance and shellcode the source datasets have 112,387 and 129,531 training samples, respectively. All these experiments are performed for target datasets containing 100, 200, 500, 1000, 5000 and 10000 samples, except in the case of attack category shellcode where we just go upto 2000 samples as the UNSW-NB15 dataset contains a maximum of 2600 training samples for the shellcode category.
Experiments: We use our adversarial DA approach by training the GAN mentioned in Section 4.2 using the source and target datasets and then use the generator of the GAN as a classification model for testing. For our approach, the pre-processing techniques transform the source and datasets into datasets having 30 features each before using adversarial DA.
Performance results on the target dataset: Fig. 6 reports the accuracy and f-scores for this scenario for all three attack categories: exploits, reconnaissance, and shellcode. The x-axis represents the number of target dataset samples used for training and the y-axis represents the accuracy and f-scores when the classification models are tested on the target testing dataset. For the attack category exploits, we observe that our adversarial DA approach outperforms the base case and the fine-tuning approach for 100 target dataset samples by 7.66% and 6.73% respectively in terms of accuracy and

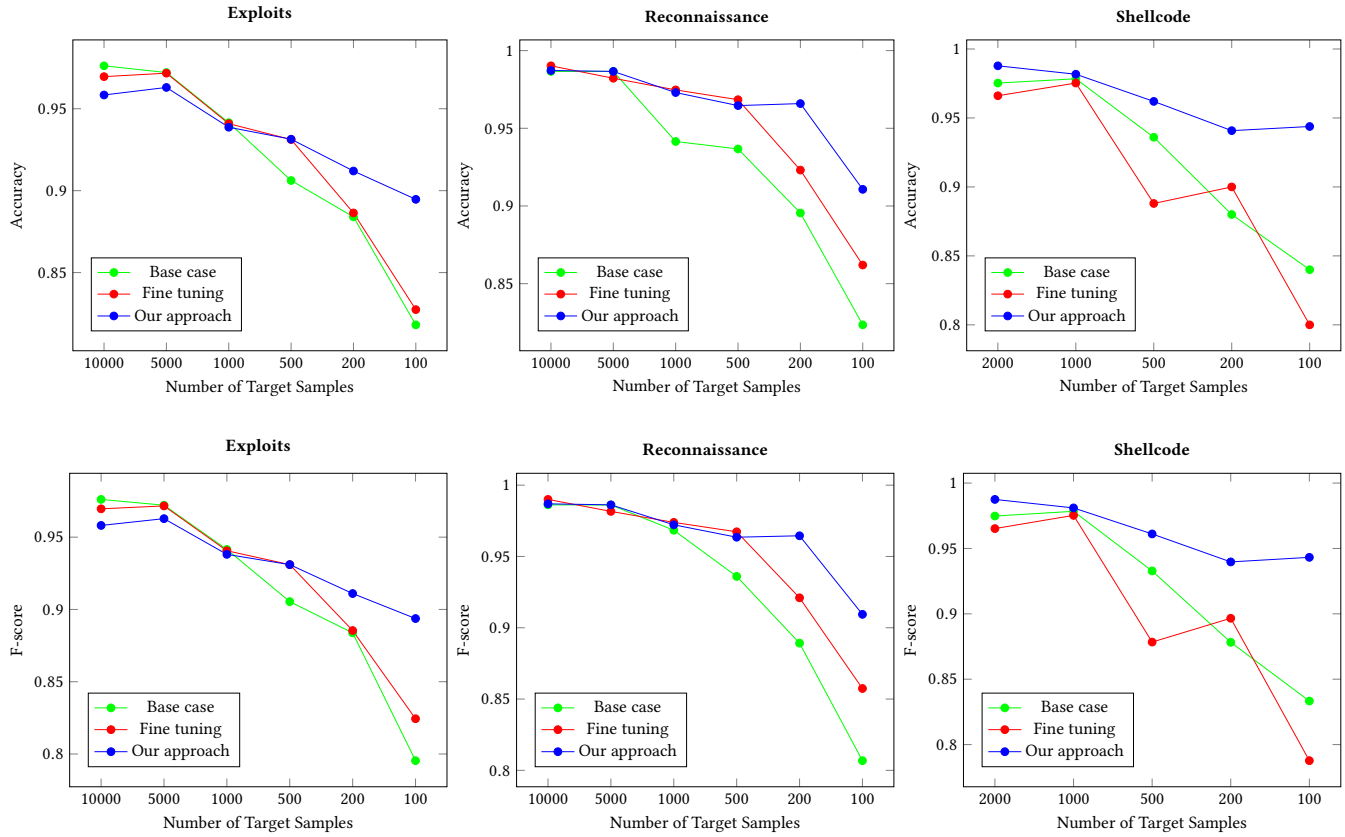

Figure 6: Accuracy and F-score results on the target dataset for the scenario where source and target datasets have same feature spaces comparing three approaches: base model trained with only the target dataset samples; model trained on the source dataset and fine-tuned on the target dataset; model trained using our adversarial DA approach.

Samples in target dataset used for training.	Accuracy on source dataset					
	Exploits		Reconnaissance		Shellcode	
	Fine-tuning	Our approach	Fine-tuning	Our approach	Fine-tuning	Our approach
100	0.8395	0.9134	0.7566	0.9388	0.5955	0.9281
200	0.851	0.9459	0.616	0.9246	0.8365	0.9529
500	0.7477	0.9484	0.7474	0.9017	0.6979	0.9294
1000	0.6077	0.9301	0.5354	0.9052	0.5842	0.9292
5000	0.7453	0.9423	0.578	0.9181	-	-
10000	0.7777	0.9489	0.7626	0.9301	-	-

Table 3: Performance of the NID models trained using fine-tuning approach and our approach on the source dataset. We compare our approach with the fine-tuning transfer learning approach.

by 9.83% and 6.93% in terms of F-score. We obtain similar results from the experiments with other attack categories reconnaissance and shellcode, where our approach outperforms the other two comparison approaches in terms of classification accuracy and f-scores, for less than 500-1000 samples.

The low accuracy of the other approaches when trained with a small amount of data samples is a result of extreme over-fitting often seen when DL models are trained with very little training data. Small amount of training data makes DL classification models really good at classifying the training data but cannot generalize well on unseen data. Our adversarial DA approach essentially uses the source dataset samples to augment the target dataset and projects them into a common latent subspace, thus, alleviating the problem of over-fitting due to very small amounts of training data.

Performance on the source dataset: We also test the accuracy of the resulting classification models from the transfer learning using fine-tuning approach and our adversarial DA approach on the source dataset. This would represent the models' performance

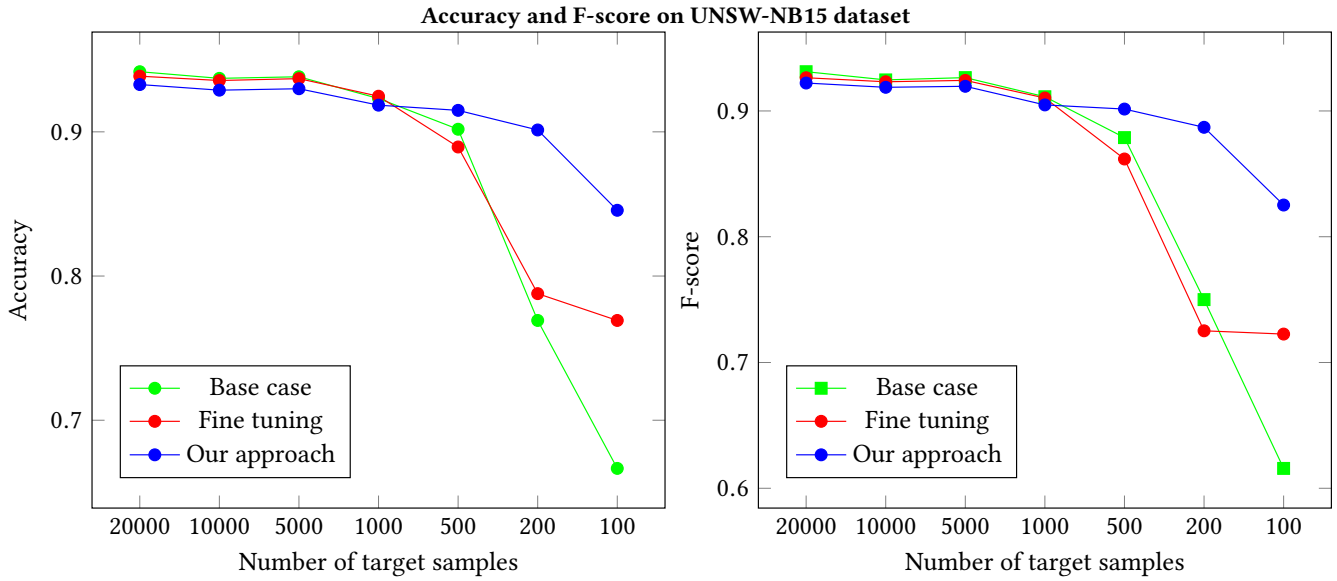

Figure 7: Accuracy and F-score results on the target dataset for the scenario where source and target datasets have different feature spaces comparing three different approaches: base model trained with only the target dataset samples; model trained on the source dataset and fine-tuned on the target dataset; model trained using our adversarial DA approach.

on the old source dataset with 8 attack categories. We do not include the base case where the model is just trained on the target dataset as it performs really badly in classifying the source attack categories. We report the accuracy results for the classification models on the source dataset in Table 3. We observe that our adversarial DA approach maintains an accuracy rate of 91-94% on the source data, whereas the model created using the fine-tuning approach is highly unsteady. This demonstrates that our approach performs well not only with respect to the target dataset but is also highly accurate with respect to source datasets allowing it to detect the new attacks seen in the target dataset as well as the old attacks seen in the source dataset.

Combining the source and target datasets: One naive approach to train a NID model that can identify attacks contained in both the source dataset and the target dataset would be to just combine the two datasets and use the combined dataset for training. To assess such a naive approach, we have carried out additional experiments. The results from such experiments show that, although the NID models trained using this approach achieve high accuracy and f-score on the source dataset, they perform worse than our adversarial DA approach when the target dataset has very low numbers of training samples. When trained with the combination of the source dataset and the target dataset with 100 samples for exploits, reconnaissance and shellcode, the adversarial DA approach outperforms this naive approach by 6.03%, 8.26% and 3.45% respectively, in terms of accuracy on the target dataset.

6.2 Different feature space

This category represents the scenario where the target dataset has a different feature space than the source dataset.

Datasets: For this evaluation, we use NSL-KDD [46] as the source dataset and UNSW-NB15 [34] as the target dataset. We remove the specific categories of attacks from the datasets and model this as a binary classification problem, i.e. predicting whether the current record belongs to the attack or benign category. In this evaluation, the source NSL-KDD dataset has 62,986 labeled training samples and we vary the target UNSW-NB15 dataset samples used for training. We perform experiments for 100, 200, 500, 1000, 5000, 10000 and 20000 labeled training samples from the target dataset.

Experiments: Since the fine-tuning approach only works if the source and the target datasets have the same feature space, we apply the pre-processing techniques mentioned in Section 4.1 to both datasets to transform them into datasets with 30 features each before using the fine-tuning approach. We use our adversarial DA approach by training the GAN described in Section 4.2 using the NSL-KDD (source) and the UNSW-NB15 (target) datasets and then use the generator of the GAN as a classification model for testing. Similar to the fine-tuning case, our pre-processing techniques transform the source and datasets into datasets having 30 features each before using adversarial DA.

Results: Fig. 7 presents the results for this scenario. From our experiments, we observe that our adversarial DA approach outperforms the base case and the fine-tuning approach for 100 target dataset samples by 17.9% and 5.78% respectively in terms of accuracy and by 20.94% and 10.26% in terms of F-score. The fine-tuning approach improves the performance over the base case but still performs worse than our approach. This is also a result of extreme over-fitting often seen when DL models are trained with very little training data. These results are similar to the results for experiments with source and target datasets with same features allowing us to

conclude that our adversarial DA approach performs equally well when the source and target datasets have different feature spaces.

Similar to the case where source and target datasets have similar feature spaces, the NID model trained our adversarial DA approach outperforms the base case and fine tuning case when tested on the source dataset. However, in most of the scenarios where the source dataset has a different feature space than the target dataset, we might not be interested in the efficacy of the trained NID model in identifying the attacks contained in the source dataset. So we omit the results from the paper.

7 CONCLUSION AND FUTURE WORK

In this paper, we propose an adversarial DA approach to train DL classification models for NID with very small amount of labeled training data. This allows organizations to train NID models to identify new families of attacks on their networks by leveraging existing datasets, for example publicly available ones or datasets used in the past, and just capturing a small amount of labeled data for the new attack families. Our experiments show that the NID models trained using our adversarial DA approach outperform other approaches, like transfer learning using fine-tuning, both when the source and the target dataset have similar features (homogeneous DA) and when the two datasets have different features (heterogeneous DA).

As future work, we plan to consider the cases of semi-supervised DA (some labeled samples and a lot of unlabeled samples in the target dataset) and unsupervised DA (only unlabeled samples in the target dataset). We also plan to evaluate more sophisticated GAN architectures like Wasserstein GANs [3]. Furthermore, in this paper we model the NID problem as a binary classification problem, i.e. identify whether the current data sample is an attack sample or a benign sample. However, it would be interesting to explore the efficacy of adversarial DA approaches for multi-class classification i.e. identifying the specific categories of attack for the target datasets. One can also evaluate the efficacy of the adversarial DA approach by using target datasets containing labeled data only for specific networks like IoT networks or mobile networks. In addition to this, adversarial DA can also be evaluated using more than one source domains. This might result in a more generalized model able to identify attacks for multiple network types. Also, our adversarial DA approach requires the source dataset as well as the target dataset for training NID models. However, it may be difficult to meet such requirement when the organization that collected the source dataset is different from the target organization and the former would like to keep its source dataset private while at the same time helping the target organization to enhance its NID functions. Such a scenario can be considered an example of federated learning. To address such issues, privacy preserving DA techniques need to be developed so that the source dataset can be used for DA, but without having to disclose the source dataset.

8 ACKNOWLEDGEMENTS

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

REFERENCES

[1] [n.d.]. NSL-KDD dataset. https://www.unb.ca/cic/datasets/nsl.html.
[2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. 2016. TensorFlow: A System for Large-Scale Machine Learning.. In *OSDI*, Vol. 16. 265–283.
[3] Martin Arjovsky, Soumith Chintala, and Léon Bottou. 2017. Wasserstein Generative Adversarial Networks. In *International Conference on Machine Learning*. 214–223.
[4] Konstantinos Bousmalis, Alex Irpan, Paul Wohlhart, Yunfei Bai, Matthew Kelcey, Mrinal Kalakrishnan, Laura Downs, Julian Ibarz, Peter Pastor, Kurt Konolige, et al. 2018. Using simulation and domain adaptation to improve efficiency of deep robotic grasping. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 4243–4250.
[5] Konstantinos Bousmalis, George Trigeorgis, Nathan Silberman, Dilip Krishnan, and Dumitru Erhan. 2016. Domain separation networks. In *Advances in neural information processing systems*. 343–351.
[6] James Cannady. 1998. Artificial neural networks for misuse detection. In *National information systems security conference*, Vol. 26. Baltimore.
[7] F. Chollet. 2015. Keras. *GitHub* (2015). https://github.com/fchollet/keras.
[8] Wenyuan Dai, Gui-Rong Xue, Qiang Yang, and Yong Yu. 2007. Transferring naive bayes classifiers for text classification. In *AAAI*, Vol. 7. 540–545.
[9] Jun Deng, Zixing Zhang, Erik Marchi, and Bjorn Schuller. 2013. Sparse autoencoder-based feature transfer learning for speech emotion recognition. In *2013 Humaine Association Conference on Affective Computing and Intelligent Interaction*. IEEE, 511–516.
[10] Chuong B Do and Andrew Y Ng. 2006. Transfer learning for text classification. In *Advances in Neural Information Processing Systems*. 299–306.
[11] Maayan Frid-Adar, Idit Diamant, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. 2018. GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing* 321 (2018), 321–331.
[12] Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. 2018. Synthetic data augmentation using GAN for improved liver lesion classification. In *2018 IEEE 15th international symposium on biomedical imaging (ISBI 2018)*. IEEE, 289–293.
[13] Yaroslav Ganin, Evgeniya Ustinova, Hana Ajakan, Pascal Germain, Hugo Larochelle, François Laviolette, Mario Marchand, and Victor Lempitsky. 2016. Domain-adversarial training of neural networks. *The Journal of Machine Learning Research* 17, 1 (2016), 2096–2030.
[14] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. 2015. Domain generalization for object recognition with multi-task autoencoders. In *Proceedings of the IEEE international conference on computer vision*. 2551–2559.
[15] Muhammad Ghifary, W Bastiaan Kleijn, Mengjie Zhang, David Balduzzi, and Wen Li. 2016. Deep reconstruction-classification networks for unsupervised domain adaptation. In *European Conference on Computer Vision*. Springer, 597–613.
[16] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In *Advances in neural information processing systems*. 2672–2680.
[17] Arthur Gretton, Karsten Borgwardt, Malte Rasch, Bernhard Schölkopf, and Alex J Smola. 2007. A kernel method for the two-sample-problem. In *Advances in neural information processing systems*. 513–520.
[18] Elike Hodo, Xavier Bellekens, Andrew Hamilton, Pierre-Louis Dubouilh, Ephraim Iorkyase, Christos Tachtatzis, and Robert Atkinson. 2016. Threat analysis of IoT networks using artificial neural network intrusion detection system. In *Networks, Computers and Communications (ISNCC), 2016 International Symposium on*. IEEE, 1–6.
[19] Judy Hoffman, Eric Tzeng, Taesung Park, Jun-Yan Zhu, Phillip Isola, Kate Saenko, Alexei A Efros, and Trevor Darrell. 2017. Cycada: Cycle-consistent adversarial domain adaptation. *arXiv preprint arXiv:1711.03213* (2017).
[20] Jui-Ting Huang, Jinyu Li, Dong Yu, Li Deng, and Yifan Gong. 2013. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE, 7304–7308.
[21] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. 2017. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1125–1134.
[22] IXIA. 2019. IXIA PerfectStorm. https://www.ixiacom.com/products/perfectstorm

[23] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. 2018. Progressive Growing of GANs for Improved Quality, Stability, and Variation. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=Hk99zCeAb

[24] Georgios Kathareios, Andreea Anghel, Akos Mate, Rolf Clauberg, and Mitch Gusat. 2017. Catch It If You Can: Real-Time Network Anomaly Detection with Low False Alarm Rates. In *Machine Learning and Applications (ICMLA), 2017 16th IEEE International Conference on*. IEEE, 924–929.

[25] Diederik P. Kingma and Jimmy Ba. 2014. Adam: A Method for Stochastic Optimization. *CoRR* abs/1412.6980 (2014). arXiv:1412.6980 http://arxiv.org/abs/1412.6980

[26] Alex Krizhevsky, Geoffrey Hinton, et al. 2009. *Learning multiple layers of features from tiny images*. Technical Report. Citeseer.

[27] MIT Lincoln Labs. 1999. KDD Cup 1999 Data. http://kdd.ics.uci.edu/databases/kddcup99/task.html

[28] Yann LeCun. 1998. The MNIST database of handwritten digits. *http://yann. lecun. com/exdb/mnist/* (1998).

[29] Ming-Yu Liu and Oncel Tuzel. 2016. Coupled generative adversarial networks. In *Advances in neural information processing systems*. 469–477.

[30] Matthew V Mahoney and Philip K Chan. 2003. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *International Workshop on Recent Advances in Intrusion Detection*. Springer, 220–237.

[31] Giovanni Mariani, Florian Scheidegger, Roxana Istrate, Costas Bekas, and Cristiano Malossi. 2018. Bagan: Data augmentation with balancing GAN. *arXiv preprint arXiv:1803.09655* (2018).

[32] John McHugh. 2000. Testing intrusion detection systems: a critique of the 1998 and 1999 darpa intrusion detection system evaluations as performed by lincoln laboratory. *ACM Transactions on Information and System Security (TISSEC)* 3, 4 (2000), 262–294.

[33] Zhong Meng, Jinyu Li, Zhuo Chen, Yang Zhao, Vadim Mazalov, Yifan Gang, and Biing-Hwang Juang. 2018. Speaker-invariant training via adversarial learning. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 5969–5973.

[34] Nour Moustafa and Jill Slay. 2015. UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set). In *Military Communications and Information Systems Conference (MilCIS), 2015*. IEEE, 1–6.

[35] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. 2014. Learning and transferring mid-level image representations using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1717–1724.

[36] Chandrika Palagiri. 2002. Network-based intrusion detection using neural networks. *Department of Computer Science Rensselaer Polytechnic Institute Troy, New York* (2002), 12180–3590.

[37] Sinno Jialin Pan and Qiang Yang. 2010. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering* 22, 10 (2010), 1345–1359.

[38] Santiago Pascual, Antonio Bonafonte, and Joan Serrà. 2017. SEGAN: Speech Enhancement Generative Adversarial Network. *Proc. Interspeech 2017* (2017), 3642–3646.

[39] Karl Pearson. 1901. LIII. On lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 2, 11 (1901), 559–572.

[40] Victor Powell. 2015. Principal Component Analysis explained visually. http://setosa.io/ev/principal-component-analysis/

[41] Rajat Raina, Alexis Battle, Honglak Lee, Benjamin Packer, and Andrew Y Ng. 2007. Self-taught learning: transfer learning from unlabeled data. In *Proceedings of the 24th international conference on Machine learning*. ACM, 759–766.

[42] Markus Ring, Sarah Wunderlich, Deniz Scheuring, Dieter Landes, and Andreas Hotho. 2019. A survey of network-based intrusion detection data sets. *Computers & Security* (2019).

[43] Ankush Singla, Elisa Bertino, and Dinesh Verma. 2019. Overcoming the Lack of Labeled Data: Training Intrusion Detection Models Using Transfer Learning. In *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*. IEEE, 69–74.

[44] Baochen Sun and Kate Saenko. 2016. Deep coral: Correlation alignment for deep domain adaptation. In *European Conference on Computer Vision*. Springer, 443–450.

[45] Yaniv Taigman, Adam Polyak, and Lior Wolf. 2016. Unsupervised cross-domain image generation. *arXiv preprint arXiv:1611.02200* (2016).

[46] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. 2009. A detailed analysis of the KDD CUP 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*. IEEE, 1–6.

[47] Aaron Tuor, Samuel Kaplan, Brian Hutchinson, Nicole Nichols, and Sean Robinson. 2017. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. In *Workshops at the Thirty-First AAAI Conference on Artificial Intelligence*.

[48] Eric Tzeng, Judy Hoffman, Trevor Darrell, and Kate Saenko. 2015. Simultaneous deep transfer across domains and tasks. In *Proceedings of the IEEE International Conference on Computer Vision*. 4068–4076.

[49] Eric Tzeng, Judy Hoffman, Kate Saenko, and Trevor Darrell. 2017. Adversarial discriminative domain adaptation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7167–7176.

[50] Marcel van Gerven and Sander M. Bohte. 2017. Editorial: Artificial Neural Networks as Models of Neural Information Processing. *Front. Comput. Neurosci.* 2017 (2017). https://doi.org/10.3389/fncom.2017.00114

[51] Mei Wang and Weihong Deng. 2018. Deep visual domain adaptation: A survey. *Neurocomputing* 312 (2018), 135–153.

[52] Svante Wold, Kim Esbensen, and Paul Geladi. 1987. Principal component analysis. *Chemometrics and intelligent laboratory systems* 2, 1-3 (1987), 37–52.

[53] Lantao Yu, Weinan Zhang, Jun Wang, and Yong Yu. 2017. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*.

[54] Juan Zhao, Sachin Shetty, and Jan Wei Pan. 2017. Feature-based transfer learning for network security. In *MILCOM 2017-2017 IEEE Military Communications Conference (MILCOM)*. IEEE, 17–22.

[55] Juan Zhao, Sachin Shetty, Jan Wei Pan, Charles Kamhoua, and Kevin Kwiat. 2019. Transfer learning for detecting unknown network attacks. *EURASIP Journal on Information Security* 2019, 1 (2019), 1.

Table 4: Features in the UNSW-NB15 dataset. Src: [34]

	Name	Type	Description
1	state	discrete	Indicates the state and its dependent protocol
2	dur	continuous	Record total duration
3	sbytes	continuous	Source to destination transaction bytes
4	dbytes	continuous	Destination to source transaction bytes
5	sttl	continuous	Source to destination time to live value
6	dttl	continuous	Destination to source time to live value
7	sloss	continuous	Source packets re-transmitted or dropped
8	dloss	continuous	Destination packets re-transmitted or dropped
9	service	discrete	http, ftp, smtp, ssh, dns, ftp-data ,irc and (-) if no service used.
10	sload	continuous	Source bits per second
11	dload	continuous	Destination bits per second
12	spkts	continuous	Source to destination packet count
13	dpkts	continuous	Destination to source packet count
14	swin	continuous	Source TCP window advertisement value
15	dwin	continuous	Destination TCP window advertisement value
16	stcpb	continuous	Source TCP base sequence number
17	dtcpb	continuous	Destination TCP base sequence number
18	smeansz	continuous	Mean of the packet size transmitted by the src
19	dmeansz	continuous	Mean of the packet size transmitted by the dst
20	trans_depth	continuous	Represents the pipelined depth into the connection of http request/response transaction
21	res_bdy_len	continuous	Actual uncompressed content size of the data transferred from the server's http service.
22	sjit	continuous	Source jitter (mSec)
23	djit	continuous	Destination jitter (mSec)
24	stime	continuous	Record start time
25	ltime	continuous	Record last time
26	sintpkt	continuous	Source interpacket arrival time (mSec)
27	dintpkt	continuous	Destination interpacket arrival time (mSec)
28	tcprtt	continuous	TCP connection setup time i.e. the time between the SYN and the SYN_ACK packets.
29	synack	continuous	TCP connection setup time i.e. the time between the SYN and the SYN_ACK packets.
30	ackdat	continuous	TCP connection setup time the time between the SYN_ACK and the ACK packets.
31	is_sm_ips_ports	discrete	If source (1) and dest (3)IP addrs equal and port numbers (2)(4) equal, variable 1 else 0
32	ct_state_ttl	continuous	No. for each state (6) according to range of values for source/destination ttl (10) (11).
33	ct_flw_http_mthd	continuous	No. of flows that has methods such as Get and Post in http service.
34	is_ftp_login	discrete	If the ftp session is accessed by user and password then 1 else 0.
35	ct_ftp_cmd	continuous	No of flows that has a command in ftp session.
36	ct_srv_src	continuous	No. of connections that contain the same service (14) and srcip (1) in 100 connections (26).
37	ct_srv_dst	continuous	No. of connections that contain the same service (14) and dstip (3) in 100 connections (26).
38	ct_dst_ltm	continuous	No. of connections of the same destination address (3) in 100 connections (26).
39	ct_src_ ltm	continuous	No. of connections of the same source address (1) in 100 connections (26).
40	ct_src_dport_ltm	continuous	No of connections of the same srcip (1) and the dsport (4) in 100 connections (26).
41	ct_dst_sport_ltm	continuous	No of connections of the same dstip (3) and the source port (2) in 100 connections (26).
42	ct_dst_src_ltm	continuous	No of connections of the same srcip (1) and dstip (3) address in in 100 connections (26).
43	attack_cat	discrete	The name of each attack category. This data set contains nine categories
44	Label	discrete	0 for normal and 1 for attack records

Table 5: Features in the NSL-KDD dataset. Src: [27]

	Feature name	Type	Description
1	duration	continuous	Length (number of seconds) of the connection
2	protocol_type	discrete	Type of the protocol, e.g. tcp, udp, etc.
3	service	discrete	Network service on the destination, e.g., http, telnet, etc.
4	src_bytes	continuous	Number of data bytes from source to destination
5	dst_bytes	continuous	Number of data bytes from destination to source
6	flag	discrete	Normal or error status of the connection
7	land	discrete	1 if connection is from/to the same host/port; 0 otherwise
8	wrong_fragment	continuous	Number of "wrong" fragments
9	urgent	continuous	Number of urgent packets
10	hot	continuous	Number of "hot" indicators
11	num_failed_logins	continuous	Number of failed login attempts
12	logged_in	discrete	1 if successfully logged in; 0 otherwise
13	num_compromised	continuous	Number of "compromised" conditions
14	root_shell	discrete	1 if root shell is obtained; 0 otherwise
15	su_attempted	discrete	1 if "su root" command attempted; 0 otherwise
16	num_root	continuous	Number of "root" accesses
17	num_file_creations	continuous	Number of file creation operations
18	num_shells	continuous	Number of shell prompts
19	num_access_files	continuous	Number of operations on access control files
20	num_outbound_cmds	continuous	Number of outbound commands in an ftp session
21	is_hot_login	discrete	1 if the login belongs to the "hot" list; 0 otherwise
22	is_guest_login	discrete	1 if the login is a "guest"login; 0 otherwise
23	count	continuous	Number of connections to the same destination host as the current connection in the past two seconds
24	serror_rate	continuous	% of connections that have "SYN" errors among the connections aggregated in (23)
25	rerror_rate	continuous	% of connections that have "REJ" errors among the connections aggregated in (23)
26	same_srv_rate	continuous	% of connections to the same service among the connections aggregated in (23)
27	diff_srv_rate	continuous	% of connections to different services among the connections aggregated in (23)
28	srv_count	continuous	Number of connections to the same service as the current connection in the past two seconds
29	srv_serror_rate	continuous	% of connections that have "SYN" errors among the connections aggregated in (28)
30	srv_rerror_rate	continuous	% of connections that have "REJ" errors among the connections aggregated in (28)
31	srv_diff_host_rate	continuous	% of connections to different hosts among the connections aggregated in (28)
32	dst_host_count	continuous	Number of connections to the same destination host as the current connection in the past two seconds
33	dst_host_serror_rate	continuous	% of connections that have "SYN" errors among the connections aggregated in (32)
34	dst_host_rerror_rate	continuous	% of connections that have "REJ" errors among the connections aggregated in (32)
35	dst_host_same_srv_rate	continuous	% of connections to the same service among the connections aggregated in (32)
36	dst_host_diff_srv_rate	continuous	% of connections to different services among the connections aggregated in (32)
37	dst_host_srv_count	continuous	Number of connections to the same service as the current connection in the past two seconds
38	dst_host_same_src_port_rate	continuous	The percentage of connections that were to the same source port, among the connections aggregated in (33)
39	dst_host_srv_serror_rate	continuous	% of connections that have "SYN" errors among the connections aggregated in (33)
40	dst_host_srv_rerror_rate	continuous	% of connections that have "REJ" errors among the connections aggregated in (33)
41	dst_host_srv_diff_host_rate	continuous	% of connections to different hosts among the connections aggregated in (33)
42	label	discrete	Attack or normal