Learning Realistic Patterns from Unrealistic Stimuli: Generalization and Data Anonymization

Konstantinos Nikolaidis∗1, Stein Kristiansen1, Thomas Plagemann1, Vera Goebel1, Knut Liestl1, Mohan Kankanhalli2, Gunn Marit Traaen3,4,5, Britt verland6, Harriet Akre4,10, Lars Aakery7,8, and Sigurd Steinshamn7,8

1Department of Informatics, University of Oslo, Norway
2Department of Computer Science, National University of Singapore, Singapore
3Department of Cardiology, Oslo University Hospital, Rikshospitalet, Oslo, Norway
4Institute of Clinical Medicine, Faculty of Medicine, University of Oslo, Oslo, Norway
5KG Jebsen Center for Cardiac Research, University of Oslo, Norway and Center for Heart Failure Research, Oslo University Hospital, Norway
6Department of Otorhinolaryngology, Head & Neck Surgery, Sleep Unit, Lovisenberg Diakonale Hospital, Oslo, Norway
7Department of Thoracic Medicine, St. Olavs University Hospital, Trondheim, Norway
8KG Jebsen Center of Exercise in Medicine, Department of Circulation and Medical Imaging, Faculty of Medicine and Health Science, Norwegian University of Science and Technology, Trondheim, Norway.
9Department of Cardiology and Center for Cardiological Innovation, Oslo University Hospital, Rikshospitalet, Oslo, Norway
10Department of Otorhinolaryngology, Head & Neck Surgery, Oslo University Hospital, Rikshospitalet, Oslo, Norway.

Abstract

Good training data is a prerequisite to develop useful ML applications. However, in many domains existing data sets cannot be shared due to privacy regulations (e.g., from medical studies). This work investigates a simple yet unconventional approach for anonymized data synthesis to enable third parties to benefit from such private data. We explore the feasibility of learning implicitly from unrealistic, task-relevant stimuli, which are synthesized by exciting the neurons of a trained deep neural network (DNN). As such, neuronal excitation serves as a pseudo-generative model. The stimuli data is used to train new classification models. Furthermore, we extend this framework to inhibit representations that are associated with specific individuals. We use sleep monitoring data from both an open and a large closed clinical study and evaluate whether (1) end-users can create and successfully use customized classification models for sleep apnea detection, and (2) the identity of participants in the study is protected. Extensive comparative empirical investigation shows that different algorithms trained on the stimuli are able generalize successfully on the same task as the original model. However, architectural and algorithmic

∗Corresponding Author: konstan@ifi.uio.no
similarity between new and original models play an important role in performance. For similar architectures, the performance is close to that of using the true data (e.g., Accuracy difference of 0.56%, Kappa coefficient difference of 0.03-0.04). Further experiments show that the stimuli can to a large extent successfully anonymize participants of the clinical studies.

1 Introduction

In recent years, machine learning (ML) has become a viable solution for various applications due to rapid developments in sensor technologies, data acquisition tools, and ML algorithms (e.g., deep learning). It is well-known that training data of sufficient quality and quantity is a pre-requisite to train a ML classification model (classifier), that can generalize reliably. However, there are situations in which access to data that fulfil these requirements is restricted, e.g., due to privacy concerns.

Such situations are particularly prominent in the medical domain. One example is the Cesar project [1], which aims to enable individuals to perform sleep monitoring at home with low-cost sensors and ML-based automatic sleep apnea detection on their smart-phone. Having access to labelled sensor data from a large clinical study enables us in the project to train ML classification models and evaluate their performance. The final goal of the project is to allow any individual to use a customized classifier that is tailored to the particular needs of the individual. However, regulatory restrictions prohibit us to share the data, neither with individuals so that they can create their own customized classifiers, nor for other scientific purposes. Creating for any interested individual a customized classifier in our lab is no feasible solution.

A possible solution to this problem is to release an anonymized version of the data with the use of existing database anonymization strategies like k-anonymity [2] or l-diversity [3]. However, since crucial parts of our data are raw sensory time-series data from which identification can be done indirectly via learning, existing database anonymization strategies are not suitable for this task. Another option would be to only release white-box or black-box classifiers in the form of an API or mechanisms that perform classification or extract important statistics from the data. For this purpose, differential privacy [4] is a well-established framework that offers theoretical guarantees for privacy preserving application of statistical mechanisms. Works such as [5] make differential privacy a viable option for ML applications. However, this option defeats our goal of giving customization freedom to the end-users. Additionally, though theoretically sound, differential privacy has been shown to yield in many cases unacceptable privacy-performance trade-offs [6], and to be susceptible to different information leakage attacks [7, 8]. A third option is to train a differentially private generative model like [9] or [10] on our data, in order to synthesize a dataset that we could release to the public. We avoid this option for two reasons: (1) because of the points discussed above about differential privacy; (2) as we care about the identification of recordings and not of individual datapoints, we would need to apply the
group privacy property while training such a model. However, this would either weaken the attainable privacy guarantee or the performance. Based on the above discussion, we explore in this work a different option, which is not sufficiently studied in related works. We investigate the empirical feasibility of labelled noisy higher-layer representations for training other "student" classifiers to generalize reliably on the real data. The goal is to create a labelled dataset from which a model can be taught to perform sleep apnea classification, while at the same time data from this dataset cannot be strongly identified as belonging to a specific recording. To do this, we exploit the knowledge obtained by a given trained classifier, which we refer to as Teacher, $h_T$, for notational convenience. $h_T$ is trained to capture the most important aspects of the real data based on the loss it attempts to minimize, making it learn task-related knowledge about the training data. We expect that excitatory or inhibitory datapoints (which we call stimuli) of specific neurons can also contain important information about the class decisions of $h_T$. Based on this, we learn to generate varying stimuli targeting the output of one or more $h_T$ and use these stimuli to train a student classifier $h_S$. Neuronal Excitation (NE), is a general method that can be applied on artificial neural networks [11] as well as on the mammalian inferotemporal cortex [12]. We use in this work Activation Maximization (or Minimization) (AM) [13] as NE. AM is a well-established method for interpretation of neuronal activity. The overall proposed procedure is loosely related to implicit learning [14] in the sense that for $h_S$, knowledge about features of the true joint distribution is acquired implicitly, and not through direct loss minimization on data sampled from it (or from a distribution that approximates it).

This leads us to an important novel aspect of this work that to the best of our knowledge serves as a differentiating factor compared to other generative approaches. The stimuli we are synthesizing need not necessarily be realistic. On the contrary, to some extent we want them to be unrealistic. We want $h_S$ to learn indirectly through the stimuli and generalize on the real data. Please note that such an approach, only captures those features needed to strongly excite or inhibit different class neurons of $h_T$. This is in contrast to a generative model or framework which would attempt to capture all features necessary to learn the joint or the data (marginal) distribution based on its loss. Therefore, we have more direct access to the conditional distribution we want to learn. We hypothesize that this procedure is a natural way to generate datapoints that, though unrealistic, could contain inherently important information about the class separation we care about, e.g., sleep apnea in our case. Additionally, the datapoints could potentially provide less "unwanted" information for other class separations which we want not to be learned. In this work our contributions are as follows:

- We demonstrate that the proposed approach is an empirically feasible way to learn and generalize successfully on new data from the true distribution.

- We investigate the applicability of training different smaller architectures for successful customization with the use of the generated stimuli dataset.
We compare with an existing well-established generative approach, namely gradient-penalty Wasserstein GAN [15], and illustrate promising results.

- We empirically show the viability of a variation of the proposed approach as a means of generating anonymized data. To do this we develop a patient de-anonymization attack inspired from face identification, and evaluate how the AM stimuli compare to the real data in terms of the identification success of the adversary. Furthermore, we explore the defence capability that the proposed approach offers against membership inference attacks and exhibit additional potentially useful properties of the described method.

The rest of the paper is organized as follows: Section 2 presents the proposed approach. Section 3 describes the application scenario and the datasets we use. In Section 4 we perform experiments to investigate the generalization and customization capabilities of the proposed approach. In Section 5 we investigate defensive and anonymization properties and in Section 6 we discuss additional characteristics of the proposed approach. Section 7 compares with related literature. Section 8 concludes this paper.

2 Approach

In this section we discuss the approach we use to generate a synthetic labelled dataset consisting of noisy stimuli via the use of AM.

![Diagram](image)

Figure 1: Four main steps of the proposed approach. TEACHER corresponds to \( h_T \) and STUDENT to \( h_S \)

2.1 Generating Synthetic Data with AM

We want to transfer the knowledge of a given trained DNN classifier \( h_T \) to another model or learning algorithm \( h_S \) through the use of a synthetic dataset
with \( h_S, h_T : X \rightarrow Y \), where \( X \) is the input and \( Y \) the output space. We assume that the original data \( D \) with which \( h_T \) is trained is not available for training of \( h_S \). The end-user who trains \( h_S \) has only access to \( D_S \). We aim to enable \( h_S \) to classify data that come from the same distribution as \( D \) with a similar performance as \( h_T \). One way to do this efficiently is to extract the knowledge accumulated by \( h_T \) with the creation of \( D_S \). The novelty of the proposed approach stems from the fact that we utilize AM in an unconventional manner with the goal of creating a diverse, multi-faceted dataset that can be used to train another classifier.

Depending on the success of the generation procedure to map the important features learned by \( h_T \) into \( D_S \), and the algorithmic and architectural similarity between \( h_T \) and \( h_S \), we show that it is possible for \( h_S \) to learn to perform the classification task \( h_T \) has learned.

**Design:** Our proposed design is based on four basic steps (see Figure 1):

- **Step 1** - Training of the teacher. We train \( h_T \) in a supervised manner with \( D \) to learn the underlying conditional data distribution \( p(y|x) \). This requires the original labelled training data.

- **Step 2** - Creating the synthetic dataset. We create a synthetic dataset \( D_S \) that captures features that \( h_T \) has learned from training on \( D \). A generation procedure \( Gen\{h_T\} \) is used to create this synthetic dataset. As \( Gen\{h_T\} \), we perform AM via a deep generator network (\( G_{AM} \)) that transforms a small noise vector \( z_{in} \) to examples that strongly activate a predefined neuron. Inspirations for this design are [16, 17]. After the synthetic set is created, we create its labels. If \( x_s \) is a synthetic example created by \( Gen\{h_T\} \), we give \( x_s \) the label that \( h_T \) chooses for it, i.e., either the class with the maximum output probability, \( \arg \max_i \{h_T^{(i)}(\theta,x_s)\} \) (\( i \) corresponds to the possible class of the output and \( \theta \) corresponds to the parameter vector of \( h_T \)), or the softmax of the output to capture the output probabilities of \( h_T \).

- **Step 3** - Training of the student. As next step \( h_S \) is trained by using the synthetic data and labels produced by Step 2. \( h_S \) can be a larger or smaller DNN than \( h_T \), or even be based on a different learning method, e.g., an SVM.

- **Step 4** - Use the student on the test set.

The choice of \( Gen\{h_T\} \) and which method to use for \( h_S \) are two central decisions. Next, we discuss the choice of \( Gen\{h_T\} \).

**Generation:** For the stimuli generation we consider two approaches: AM and code inversion [18], since both approaches result in expressive data based on the

\[1\] In this work we use the terms real data and original data interchangeably to describe \( D \).
acquired knowledge of \( h_T \). Code inversion requires the original training data, or the logits of the data from the fully connected layer we try to match. However, training \( h_S \) without access to \( D \) would be beneficial towards our anonymization objectives. Therefore, we use AM for the generation of the synthetic dataset. We further discuss this choice from a more theoretical viewpoint in the next section. Note that contrary to other works like \[17\], the goal of \( \text{Gen}\{h_T\} \) is not to produce realistic looking synthetic data, but instead to create a synthetic dataset that can reliably train \( h_S \).

Formally, we assume a trained model \( h_T(\theta, x) \) with \( x \) being an element of the input space \( X \), and \( \theta \) the parameter vector of the model, which is an element of parameter space \( \Theta \). To synthesize data that the trained model \( h_T(\theta, x) \) perceives as elements of a class, \( \text{Gen}\{h_T\} \) uses AM on the activations of the output layer of \( h_T(\theta, x) \) via \( G_{AM} \) such that:

\[
\theta^*_G = \arg \max_{\theta_G \in \Theta} \max_{z_{in}} \left\{ h^{(i)}_{TL}(\theta, G_{AM}(\theta_G, z_{in})) \right\} \quad \text{subject to} \quad ||G_{AM}(\theta_G, z_{in})||_{\infty} \leq R_G \tag{1}
\]

where \( i \) corresponds to the class we want to find a stimulus for, and \( \theta_G \) is the parameter vector of \( G_{AM} \). Note that \( \theta \) is static because \( h_T \) is already trained and we are optimizing for \( \theta_G \) given varying “pseudo inputs” of \( z_{in} \). \( R_G \) is the maximum allowed value of the norm of the output of \( G_{AM} \). We use the \( l_\infty \)-norm, i.e., a hyper-cube to express the bounding, and \( L \) denotes the output layer. We choose to perform AM in the output layer of \( h_T(\theta, x) \) since we want stimuli that correspond to a strong belief for a class in \( h_T \). We use \( z_{in} \) as an input random noise vector (mainly from uniform distribution) which \( G_{AM} \) transforms into the stimulus. We intentionally do not give any additional priors to \( \text{Gen}\{h_T\} \), since we do not want to constrain our exploration of the feature space.

We stop the AM when the output of the target class neuron \( i \) is higher than all the other output neurons and exceeds a threshold \( TH \):

\[
h^{(i)}_{TL}(\theta, G_{AM}(z_{in}), \theta^*_G)) > TH \tag{2}
\]

This implies that \( \theta^*_G \) that satisfies Eq. 2 exists in a subspace defined by the decision boundaries of class \( i \). We assume that softmax activation is used on the output layer.

We specify the loss of \( G_{AM} \) to maximize the logits of each class of \( h_T \) (TEACHER in Figure 1). Then, to make the generated synthetic stimuli more diverse, we add inherent randomness in the AM procedure. We do this by reinitializing the AM generator, by randomly setting \( TH \) to one of several possible values, and by editing randomly generated datapoints instead of generating them (for more details please refer to Appendix D). Our goal is to capture a wide variety of different starting positions for the gradient ascent (AM).
This means that assuming that the starting position \( x_{s,n} \) of a stimulus before the AM optimization takes place is drawn from distribution \( p_{AM,n} \), we want \( \text{supp}(p_{AM,n}) = X \), so that \( x_{s,n} \in X \). Furthermore, we want to randomize where the gradient ascent will end up, given that the threshold condition is satisfied.

We perform the AM from each initial position towards a random class with equal probability, to create a stimulus. The result is a synthetic dataset comprised of the stimuli of \( h_T \) for all the classes. The goal of the different initial feature space positions is to take advantage of the multi-faceted property of the neuronal activation [19] when performing AM to create a diverse synthetic dataset. In the next section, we analyze our choice of AM as a means to confine the feature space, in order to ”focus” \( h_S \)’s learning to important regions of the feature space.

### 2.2 Why AM?

Assuming a marginal distribution \( p_N \), with a support \( \text{supp}(p_N) = X \) from which we draw data that we label with \( h_T \) and then use them to train \( h_S \). It is straightforward to show that for \( h_T \) and \( h_S \), with the same architecture which corresponds to the same hypothesis space, i.e., \( H_S = H_T \), \( h_S \) has theoretically the ability to learn to perfectly imitate \( h_T \). Even when \( h_S \) uses a thinner variant of the network architecture of \( h_T \), it is possible to show that if certain conditions are met, we can probabilistically expect better learning behavior as we increase the hypothesis \( H_S \) towards that of \( H_T \). We defer a more detailed discussion on this topic to Appendix C.

However, despite of the above point, as the dimensionality of the input space increases, training \( h_S \) with generated data from a different marginal data distribution than the real data becomes unviable. To mitigate this effect, we confine the space in which we perform the minimization. There could be many ways to confine the space in which we wish to have low true risk. Since we have in our case no access to the true marginal distribution \( p(x) \), we confine the space to sub-regions of high confidence for our conditional estimator. Thus, one of the core ideas of this work is that, assuming that our \( h_T \) is good at approximating \( p(y|x) \), regions that the classes are inconclusive for \( h_T \) should also be inconclusive for \( p(y|x) \). Thus, by definition there cannot be a strong belief for any algorithm that estimates \( p(y|x) \) well in these sub-regions. In the next section, we explain how we use this principle to inhibit the class probabilities of a recording’s conditional approximator, i.e., a model that recognizes different recordings, for the purpose of anonymization.

### 2.3 Recording Anonymization with Inhibitory Stimuli

To satisfy our objective of generating anonymized data in relation to information about class separation for a different labelling for patient classification that we wish to hide, we need to extend our approach. Our design is loosely inspired by the analysis of [20]. We assume a new learning task \( X \rightarrow U \), different from the
original $X \to Y$. $U$ in our case is the set of all the identities of the patients we have in our dataset, which corresponds, based on some ordering of the identities, to $\{0, \ldots, N_p\} \in \mathbb{N}$ where $N_p$ is the number of patients.

To model this labelling, we use a random variable $u \in U$, conditionally dependent on $x$ by $p(u|x)$. Our goal is to generate data that are not strongly affiliated with any outcome of $u$. Formally, we want: $\forall x \in D_S, \forall u_c \in U : p(u_c|x_s) \approx p(u_c)$. We hypothesize that if $y$ and $u$ are not strongly correlated given different outcomes of $x$ then learning for $X \to Y$ would be possible with data that satisfy the previous requirement.

In practice, to achieve this, we train a network $h_{TU}$ to approximate $p(u|x)$. Then we alternate between training updates for the original objective and the following cross-entropy loss:

$$L_U(\theta_G) = - \sum_{u_c} \hat{p}(u_c) \log h_{TU}(G_{AM}(\theta_G, z_{in}))$$

The optimization is done for $\theta_G$, after training $h_T$ and $h_{TU}$. The weights of $h_T$ and $h_{TU}$ are frozen. $\hat{p}(u)$ is the empirical approximation of the marginal distribution of $u$. The optimization stops when both the threshold condition of Eq. 2 and a proximity condition regarding $\hat{p}(u)$ are satisfied (e.g., $L_U(\theta_G) < TH_U$, for some threshold $TH_U$).

3 Application Scenario and Datasets

The previous section presents a new approach to use modified noise, transformed to stimuli instead of the real data as a means of teaching new classifiers about the class separation for a task we are interested in. Furthermore, this approach can provide protection against potential adversarial information leakage attacks. These two advantages of the algorithm are individually evaluated in Section 4.
and 5. Both evaluations are performed for the application scenario of sleep apnea detection and we use for both the same datasets. Therefore, we describe in this section the application scenario and the datasets, before we (1) investigate in Section 4 how well different ML methods or architectures can generalize when trained with stimuli generated from the knowledge of $h_T$ and (2) investigate in Section 5 whether using stimuli instead of real data can provide protection for certain adversarial attacks.

In the Cesar project we want to detect Obstructive Sleep Apnea (OSA) with low-cost sensors and using ML-based analysis on smart-phones. Sleep Apnea events are defined as the cessation of airflow for at least 10 seconds or reduced airflow by at least 30% (American Academy of Sleep Medicine - AASM, [21]).

OSA can be detected by Polysomnography (PSG) in sleep laboratory or by polygraphy at home. A variety of physiological signals are commonly used for OSA detection, including the electrocardiogram (ECG), electroencephalogram (EEG), electromyogram (EMG), electrooculograph (EOG), oxygen saturation, heart rate, blood pressure and respiration from the abdomen, chest and nose.

OSA is a very common, yet severely under-diagnosed disorder. In Norway, it is estimated that around 25% of all middle-aged Norwegians are at high risk of having obstructive sleep apnea, yet approximately 70-80% of all cases are expected to be undiagnosed [22]. In our work, we use data from a large clinical study, called A3 study, at the Oslo University Hospital and St. Olavs University Hospital. In this study, sleep monitoring data from several hundred patients is collected and analyzed. As such, this is data that is collected every day in clinical settings to address a severe health issue. However, privacy regulations do not permit to share the data and prevent reproducibility of the results in this paper gained with the A3 data. Therefore, we use in addition to the A3 data the well-known open sleep monitoring data set called Apnea-ECG [23].

Sleep monitoring data are for most humans hard to evaluate and can contain noise and artifacts. Thus, we start our investigation with a simpler problem and a well understood dataset, namely digit recognition with the MNIST dataset. The insights gained with MNIST help to properly experiment and interpret results with the more challenging sleep monitoring datasets. Furthermore, the use of two different types of data indicates the generalizability of the proposed approach.

We use three datasets to evaluate our approach:

- **MNIST** is a well-known database containing a training set of 60000 28\times28 black and white images of 0-9 handwritten digits and a test set of 10000 digits. Out of the 60000 training data we use 5000 as validation set.

- **Apnea-ECG** is a well-known open database from Physionet which contains data from multiple sensors capturing ECG, respiration from the chest and abdomen, nasal airflow (NAF), and oxygen saturation. The data contain sleep recordings from 8 patients with durations of 7-10 hours. Apnea-ECG has been collected in a sleep laboratory with PSG and preprocessed such that it contains only high quality data. From Apnea-ECG we mainly use the NAF signal, because it can be used adequately to train a classifier.
to recognize apneas and yields the best single signal performance among all the respiratory signals as shown in our previous work [24]. We use the eight sleep recordings that contain the NAF signal (i.e., a01, a02, a03, a04, c01, c02, c03, b01). The sampling frequency of the sensors is 100 Hz and all recordings contain labels for every minute window of breathing that signifies whether the minute is apneic or not, i.e., whether the person is experiencing an apneic event during this minute or not.

- The A3 study [25] investigates the prevalence, characteristics, risk factors and type of sleep apnea in patients with paroxysmal atrial fibrillation. The data were obtained with the use of the Nox T3 sleep monitor [26] with mobile sleep monitoring at home, which in turn results in lower data quality than data from PSG in sleep-laboratories. An experienced sleep specialist scored the recordings manually using Noxturnal software such that the beginning and end of all types of apnea events is marked in the time-series data. To use the data for the experiments in this paper, we labeled every 60 second window of the data as apneic (if an apneic event happened during this time window) or as non-apneic. The data we use in the experiments is from 438 patients and comprises 241350 minutes of sleep monitoring data. The ratio of apneic to non-apneic windows is 0.238. We use the NAF signal from the A3 data in the experiments, i.e., the same signal we use from Apnea-ECG.

4 Generalization and Customization

In this section we investigate how well can models trained on $D_S$ generalize, and how feasible it is for different architectures or methods to learn from $D_S$.

4.1 Experimental Set-up

We evaluate the generalization capability of the proposed approach when $h_S$ is either similar or dissimilar to $h_T$, and we compare with other methods. For this comparison we select the gradient-penalty Wasserstein GAN [15, 27], which we will refer to as WGAN for brevity. We made this selection since this framework is very well-established (in terms of citations and github repository projects), and it is a more modern, general approach of the successful GAN framework. Additionally, the WGAN is especially stable during training and exceptionally good at avoiding mode-collapse. Furthermore, it is able to produce very realistic and stable results for our tasks. In this section we evaluate the proposed design from Section 2.1 with all three datasets described in Section 3.

We preprocess the data for each experiment. For MNIST, we rescale the data from 0-255 to 0-1. We downsample Apnea-ECG and A3 to 1Hz. We additionally rebalance the A3 dataset by subsampling randomly the majority class. As a result, the rebalanced dataset includes an equal amount of apneic and non-apneic labels. In all experiments we perform four steps: (1) train $h_T$, (2) use AM or WGAN to generate a synthetic dataset (3) train $h_S$ with the
synthetic data, and (4) evaluate $h_S$ with the test set. For MNIST we use the original test set. For Apnea-ECG and A3, we create a test set by randomly sampling from the datasets. We use 15% of the Apnea-ECG data and 20% of the A3 data as test set. For the WGAN model, we use a conditional WGAN based on [28].

Different configurations for the AM- and WGAN-based set-ups can yield very different results. Therefore, we perform in a pre-study Steps (2) to (4) with different hyperparameter configurations. We then choose the configuration that yields the best average performance of $h_S$ across three iterations.

With the configuration chosen from the pre-study we repeat Steps (2)-(4) to perform the actual test. With the selected configurations, we create the synthetic dataset in Step (2). With this dataset we evaluate the performance of $h_S$ on the test set by training and testing $h_S$ for five iterations of Steps (3,4). We use a validation set and save periodically the parameters of $h_S$. We choose the parameters which yield the best validation performance and use this model to classify the test set.

For the WGAN comparison, we assume that the host releases synthetic data, which are synthesized from a WGAN generator, in order to train $h_S$. For this comparison to be on equal terms, we design the WGAN such that the WGAN generator has a similar architectural configuration as $h_T$. Furthermore, we train $h_S$ with the original training data and evaluate it on the test set in order to understand the impact of similarity (in terms of architecture and size) onto the performance of $h_S$.

We investigate five different $h_S$ architectures, called ID, S, VS, L, and LL.
ID is identical to \( h_T \). S is similar to \( h_T \), but with half the number of weights per layer. VS has a quarter of the weights per layer compared to \( h_T \). The architecture of L is similar to \( h_T \), but without the second to last fully connected layer. In LL, the second and third to last layers (either fully connected or convolutional) are removed. For all experiments, we aim for the WGAN generator to have a similar number of parameters and architecture to \( h_T \). For the AM case we use as labels the softmax outputs of the output neurons (soft labels). For all datasets we investigated, we did not achieve significant performance gains by utilizing larger architectures than the ones we chose as \( h_T \). Furthermore, we focus our evaluation on smaller classifiers potentially for use in a resource constrained environment. All architectures for all experiments of this section are presented in Appendix B.

4.2 Stimuli

Figure 3 shows examples of AM stimuli from MNIST and Apnea-ECG. Although the stimuli are as expected not realistic, they implicitly contain useful information about their respective classes. As such, a classifier that learns from the stimuli is able to a certain extent to generalize on the real data, depending on the algorithm used (see Section 4.6). In both cases, the stimuli can be drastically different, even when they represent the same class. This diversity is beneficial for the student’s learning. Additionally, it is hard in both cases to distinguish between the classes for the stimuli.

To verify that the synthetic data did not contain true datapoints from the original training data, we found the closest distance datapoints between the AM produced dataset and the true dataset for all the datasets we experimented with (MNIST, ApneaECG, A3). In Appendix E we present the closest neighbors for all datasets we used. In all cases, no datapoint from the training set was recreated in the synthetic data.

4.3 Digit Classification

We follow the aforementioned set-up and investigate the performance of \( h_S \). As \( h_T \) we use a convolutional neural network loosely based on LeNet-5 [29], but with one convolutional layer less and more weights per layer. Based on this, \( h_T \) comprises of two convolutional, one max-pool and two relu fully connected layers (conv2D, maxpool, conv2D, fc1, fc2, dropout on fc1). We choose this architecture since it is a well-established model for digit classification. Furthermore, we use more weights per layer to have a larger initial model as \( h_T \), since it also achieves better on average performance than the thinner variants we experimented with. Similarly, the WGAN generator is a deconvolutional generator network with two fully connected and two deconvolutional layers with a few more parameters per layer than \( h_T \). ID has the same architecture as \( h_T \). S,\footnote{The network \( h_T \) corresponds to the TEACHER of Figure 1.}

\footnote{With the exception of MNIST, where we experiment with an even smaller design. For more details please refer to Appendix B.}
Figure 4: Train (a) and Validation (b) loss for the real dataset (blue), WGAN (orange) and AM generation (Green) with MNIST. In both Figures we show the graphs from batch 150 for better scaling.

VS, L, and LL are structured as defined above. The third to last layer of LL is a convolutional layer. For more details on the different architectures, weights per layer, filter sizes, including the AM generator, please refer to Appendix B. $h_T$ is trained with a batch size of 128 for 3125 batch iterations (see Section 6). In this experiment we generate 60000 AM stimuli and WGAN data. The experiment is repeated five times.

Table 1: Accuracy of the $h_T$ for MNIST experiment, and structures of the $h_S$.
c: convolutional layer, m: maxpooling, d: fully connected. The Max. Layer column depicts the maximum number of channels and neurons in a single layer used for convolutional layers and fully connected layers respectively.

| Structure | MaxLayer | Baseline | WGAN     | AM       |
|-----------|----------|----------|----------|----------|
| ID: cmcmdd | c:32 - d:1372 | 99.20±0.02 | 97.63±0.08 | 98.64±0.24 |
| S: cmcmdd  | c:16 - d:588   | 99.11±0.02 | 97.56±0.03 | 98.19±0.06 |
| VS: cmcmdd | c:16 - d:64    | 98.73±0.13 | 97.24±0.06 | 97.45±0.11 |
| L: cmcmd   | c:32 - d:1372  | 98.89±0.04 | 97.50±0.05 | 98.04±0.05 |
| LL: cmmd   | c:32 - d:1568  | 98.68±0.04 | 97.05±0.06 | 94.86±0.12 |

Table 1 summarizes the results from the MNIST experiment. AM outperforms the similar WGAN generator for all $h_S$ architectures, except LL. The differences between the AM and WGAN results are statistically significant for the one paired t-test ($p = 5\%$) for all $h_S$ architectures, except VS. This could potentially be attributed to a better estimation of the class boundaries of the test set from the AM stimuli than from the WGAN data. Additionally, we hypothesize that $h_T$ contains more information about the classification than the WGAN generator, since its loss is directly determined by the classification goal, which is not the case for the WGAN generator. Thus, if this knowledge is sufficiently mapped on a dataset, it could represent better the class separation boundaries than a dataset from the WGAN generator which potentially only
focuses on realistic class data.

However, notice the much lower performance of $h_S$ with the LL architecture for the AM generation compared to the WGAN. Generally, as the architecture of $h_S$ becomes smaller the drop in performance is steeper for AM than for the WGAN. This is a recurring phenomenon through all our experiments and we discuss it in detail in Sections 4.5 and 4.6. The Baseline performance exceeds that of both methods in all cases, as expected.

**Training and validation loss:** We calculate the training and validation loss throughout the training of the ID model for the different datasets. We present the results in Figure 4. The training loss of the AM dataset (green) is stable and larger than the training loss for the Real (Blue) and WGAN (Orange) datasets. However, we notice for the AM case a much better and more stable behavior for the Validation set. We expect that these effects can be attributed in part to the use of soft labels. We discuss these effects in more detail in Section 4.5. Also, the validation loss of the WGAN increases faster than for the real dataset. This could be attributed to the lesser variety of the WGAN data in relation to the real data.

### 4.4 Apnea Detection

For the apnea detection experiments, we repeat the same procedure as in the previous experiment. For Apnea-ECG (AE), we use MLP instead of a CNN to investigate how the procedure changes with a different architecture as $h_T$. For the A3 dataset, we use a 1D convolutional neural network (see details in Table 2 and Appendix B). All activations are relu, and dropout is used in both datasets for the second to last fully connected layer. The generators of the WGAN for the two datasets correspond to similar sizes and architectures to $h_T$. Similar architectural conventions apply to the different $h_S$ we experimented with, based on the definition in Section 4.1.

It is important to mention that for apnea detection we use the kappa coefficient [30] as performance metric. We make this choice because the kappa coefficient better captures performance characteristics in a single metric than accuracy, as it takes into account the possibility of agreement by chance between the two annotators (real labels and predictions). We present the accuracy, specificity, and sensitivity results in Appendix A. We use for both datasets batch size of 128 to train $h_T$. With AE, we train for 3000 batch iterations and with A3 for 15000 iterations. In both datasets (AE and A3) we generate 20000 stimuli.

Table 2 summarizes the results for the apnea detection experiments. We observe that for all cases, AM outperforms the WGAN for both datasets. Contrary to all other cases, for VS the results are not significantly larger for AM for the A3 dataset when compared to WGAN. For the one tailed t-test, the p-value is: 0.34 > 0.05. The performance drop of $h_S$ from ID to VS and to LL is much smaller for the WGAN case than for the AM case. In fact, for Apnea-ECG the performance of the VS $h_S$ for the WGAN dataset is better than the performance of the ID or S $h_S$. Also notice the large performance difference between
Table 2: Kappa of the $h_S$ for AE and A3 experiments for different $h_S$ sizes when training with data generated from our method and from a conditional WGAN of similar size. The MaxLayer column depicts the maximum number of neurons and channels of the net used.

| Structure | MaxLayer | Baseline | WGAN  | AM       |
|-----------|----------|----------|-------|----------|
| AE ID:    | dddd     | d:360    | 86.90±0.20 | 72.14±0.94 | 83.90±0.37 |
| AE S:     | dddd     | d:180    | 87.39±0.50 | 69.28±0.91 | 82.51±0.61 |
| AE VS:    | dddd     | d:90     | 85.67±0.27 | 72.43±1.05 | 81.63±0.41 |
| AE L:     | ddd      | d:360    | 84.49±0.25 | 71.44±0.68 | 81.6±0.71  |
| AE LL:    | dd       | d:360    | 85.62±0.27 | 70.99±0.31 | 80.90±0.31 |
| A3 ID:    | cmcmcmdd | c:64-d:512 | 67.06±0.19 | 60.39±0.31 | 62.98±0.70 |
| A3 S:     | cmcmcmdd | c:32-d:256 | 67.29±0.20 | 59.66±0.58 | 62.95±0.60 |
| A3 VS:    | cmcmcmdd | c:16-d:128 | 66.06±0.09 | 58.95±0.64 | 59.99±1.15 |
| A3 L:     | cmcmcmdd | c:64-d:512 | 67.24±0.16 | 60.14±0.37 | 63.88±0.23 |
| A3 LL:    | cmcmcmd  | c:32-d:256 | 66.66±0.13 | 58.96±0.6  | 62.13±0.29 |

4.5 Additional Insights

In most cases we investigated, the performance of $h_S$ is superior when training with AM stimuli than when training with WGAN generated data. However, as mentioned above, as we reduce the size of $h_S$, the drop in performance is steeper for the AM stimuli case than for the WGAN data case. One potential explanation of this phenomenon relates to an architectural mismatch between $h_S$ and $h_T$ (see Section 4.6). Another explanation relates to the type of labelling. We hypothesize that the effect occurs due to complex classification bounds of AM generated stimuli. These complex bounds are potentially a result of the soft labelling. This hypothesis is generally strengthened by the fact that the training loss during the training for the MNIST ID experiment is unstable and never goes to 0. This means that even the largest network does not over-train to the AM data (see Figure 4). However, if we use hard labels, information of the beliefs of $h_T$ is being lost, and we get a steep performance drop among all $h_S$. As a result, soft labels are preferable. This relates to an additional property. From Figure 4 we see that the validation loss for the AM data is exceptionally
stable in relation to the real and WGAN data. This means that at least for this experiment, the generalization capability of the classifier trained on the AM data was not compromised by over-training. This indicates that AM generation with soft labels introduces a form of regularization during the training of $h_S$.

### 4.6 Generalization with Different Classifiers as $h_S$

An interesting point of the above results is the consistently high performance of $h_S$ with similar architectures to that of $h_T$. This raises the question whether it is possible to achieve a correspondingly high performance when using alternative architectures or classification methods as $h_S$. We examined how a large dense deep net with relu activations, a RBF kernel SVM and a Random Forest ensemble with 50 trees perform when trained with AM stimuli on the MNIST dataset. The AM performance in all cases is low. For the SVM we have: Real Data: 94.42, AM Stimuli: 39.96. For the DNN: Real Data: 98.3, AM Stimuli: 81.06. For the RF: Real Data: 96.81, AM Stimuli: 30.82. This drop in relation to the real performance could happen since the AM stimuli are less correlated to the real test data than the real training data is. Especially for the SVM and the RF, for which the drop is much steeper than with the DNN, the difference in train and test marginal distributions can have a detrimental effect on their performance. This could happen since these models are not able to learn higher layer representations from the data, but instead they classify based on low-level decisions derived directly from support vectors or comparisons of feature values. Another explanation relates to the use of the same architectures for $h_T$ and $h_S$ in our experiments. For example, it is known that the convolutional architecture imposes \textit{inductive biases} over the learning algorithm [31]. Thus, we suppose that when $h_S$ has the same type of bias (i.e., the same architecture) as $h_T$, it can better capture and interpret how the stimuli correlate to the real data than another algorithm without this prior bias.

### 5 Defenses and Additional Properties

In this section, we experimentally evaluate the protection against specific information leakage attacks that $D_S$ or models trained on $D_S$ acquire, compared to when using the real data. Furthermore, we investigate additional potentially useful properties of $D_S$.

#### 5.1 Recording Association through Generalization

We investigate an attack against anonymized open recordings. This attack is performed by recognizing, through generalization, associations among recordings that belong to the same individual. We perform a simple, preliminary demonstration of the feasibility of the proposed attack, and show how the AM generated data can offer significantly better protection against identification in comparison to the real data.
5.1.1 Threat Model

The proposed attack draws inspiration from similar face identification tasks. We assume that an adversary has access (potentially via a security breach) to sleep recordings from a group of individuals together with personal information about them like, e.g., their names. Furthermore we assume an open dataset which contains anonymized recordings (like Apnea-ECG), and that all recording data consist of sensory signals. The goal of the attack is to probabilistically determine whether an individual has contributed a sleep recording to the open anonymized dataset. To do this, one can train a classifier to distinguish between data of different individuals that belong to the breach, and generalize to data from the open dataset. If the data sufficiently captures the idiosyncrasies of the different individuals, we expect good classification performance on new data from the same person. We ideally expect the following behavior: (1) for recordings of the open dataset that belong to individuals of the breach, we expect many minutes in the correct class i.e., classified correctly by the classifier as belonging to the correct individual; (2) for recordings of the breach that do not belong to any individual of the open dataset, we expect fewer minutes classified as belonging to this class of the classifier in relation to the previous case; and (3) for recordings that belong only to the open dataset, we expect the classifier to be more "confused" during classification of windows belonging to recordings of this type. Thus, we would expect these windows to be more uniformly distributed among the classes of the classifier in relation to case (1). Note that an important requirement for this attack is that the recordings in the two datasets contain data from the same sensor types.

Figure 5: The scenario we investigate with Apnea-ECG respiratory recordings. "A" depicts the adversary, which learns the sensory representations from the breach and generalizes on the open dataset. Her goal is to identify which individuals from the breach have participated in the open dataset. A non-adversarial user "U" utilizes the open dataset for apnea detection and generalizes on a test set. As anonymized open dataset we either use the real raw data or AM stimuli.
5.1.2 Experiment

To simplify the experiment, we assume that unbeknownst to the adversary all data from the open dataset belong to the breach. Figure 5 demonstrates the experiment. We use Apnea-ECG, and utilize all four respiratory sensors for this task, because we need as much information as possible to map patterns to specific individuals. Apnea-ECG has eight respiratory recordings corresponding to different individuals. We separate each individual original recording into three equal parts, and create the "breach recordings", the "open dataset recordings" and the test set in which we evaluate the performance of the open dataset for sleep apnea detection. From the open dataset and the final test set, we randomly discard four recording parts (a02,a03,c02,b01).

The Adversary trains a neural network to be able to differentiate the data from individuals, and attempts to find associations with the recordings in the open dataset. Meanwhile, a User utilizes the open dataset to train a model for the task of sleep apnea detection and generalizes on a test set. We explore three cases: Case (1): the open dataset includes directly the raw data; Case (2): the open dataset includes AM stimuli instead of the raw data; Case (3): the open dataset includes selectively inhibitory AM stimuli based on the extension from Section 2.3 instead of raw data. For all models, we use a similar small convolutional-fully connected architecture. For more details please refer to Appendix B.

We need to evaluate two different properties in this experiment for the three cases. First, we need to evaluate the success of the Adversary. This is not trivial, and depends on what we mean by "success". In this work, for Case (1), we evaluate by calculating the performance of the adversary in classifying correctly individual minutes i.e., to which individual each minute belongs. We use the kappa statistic as the evaluation metric. For Cases (2) and (3), we do not have direct access to the classes of the stimuli. Thus, we need a different method. We approximate the stimuli classes with $h_{TU}$ i.e., a classifier which belongs to the creator of the open dataset, that is trained to differentiate between the recordings in the open dataset. Then we can calculate performance metrics also for Cases (2) and (3) with $h_{TU}$’s beliefs as true labels. We use again the kappa statistic. To gather more information, we show classification histograms of the adversary for the three cases we investigate, and the maximum output probability per class (average across all datapoints belonging to the specified class). These graphs can be used by the attacker as tools to extrapolate recording associations, so we show them for completion. Second, we need to evaluate the success of the User on the original task, i.e., sleep apnea detection for each of the three cases. To do this we calculate the performance of each of the 3 cases on the test set. Again we use the kappa statistic as the evaluation metric. All experiments are repeated ten times.

The adversarial identification kappa is: 0.766, 0.251 and 0.004 for Cases (1),(2), and (3) respectively. As expected for the last two cases the adversary has on average less success in recognizing correctly the individuals from which the datapoints indirectly occur than for Case (1). Figure 6 (a-c) depict the per-
class percentage of classified datapoints of the adversary for the three cases. We also include the true (or approximated by $h_{TU}$) class distribution (depicted as orange). The open dataset recording classes are the first four classes, i.e., classes 0-3. For Case (1) the first four classes have much higher percentages. This is not true for Cases (2) and (3), where the adversary is not able to successfully generalize to the appropriate classes. A difference between Cases (2) and (3) is that for Case (2) the percentages are much more concentrated than for Case (3), which also holds for the approximated ”true labels”. This is expected due to the inhibitory optimization with $h_{TU}$, through which we explicitly avoid the representations that $h_{TU}$ utilizes to distinguish between different recordings. Figure 6 (d-g) show the maximum output probabilities per-class. Again, for Cases (2) and (3), we do not observe strong associations to the correct recordings. However for Case (1) the classes of the open dataset are more strongly represented than the other classes. For Case (2) some outputs are missing, because the adversarial predictions did not contain any data with maximum output of this class. The kappa of the sleep apnea detection task on the test set is: 0.98, 0.903 and 0.968 for Cases (1),(2), and (3) respectively. Interestingly, Case (2) performed on-average worse than Case (3) despite the additional constraints imposed in the feature space for Case (3).

From the above analysis we observe that the proposed procedures, especially with the extension of Section 2.3 can successfully create an anonymized dataset, with relatively minor sacrifices in performance. This makes it hard for an adversary to make correct associations about specific individuals.
5.2 Student Resilience Against Membership Inference Attacks

An additional side effect of the proposed approach is the resilience of $h_S$ against certain information leakage attacks. This phenomenon occurs since $h_S$ is trained with a substitute training dataset, whose marginal distribution is different from that of the original data. In this section, we experimentally evaluate the resilience of $h_S$ against membership inference (MI) attacks.

In MI attacks the adversary tries to determine whether a given datapoint belongs to the training dataset, based on the output of $h_S$. For a classifier $h_T$ trained on the real training data, we expect a statistically different behavior on its output when presented with data that were used to train it than when presented with data that were not used to train it. This difference can be exploited by an adversary to learn whether a datapoint belongs to the training set or not. However, since we use a very different dataset from the real training dataset which also follows different distribution, we hypothesize that it is possible to negate the effect of the MI attack for $h_S$. We follow in this experiment the approach from [32], but we simplify it. Instead of using shadow models trained on synthetic data to imitate the performance of the target classifier ($h_S$), we assume that we have auxiliary information in the form of a-priori knowledge of whether data belong to the training or test set of the target model. Thus, we train a new classifier $h_{mi}$ on the output of $h_S$ to recognize whether a datapoint belongs to the training set of $h_S$ or not. We evaluate the success of the attack with the use of kappa statistic since this is a two class problem (belong to train or to the test set), and we use the Apnea-ECG and A3 datasets for evaluation. Additionally, we apply the attack to $h_T$. Thus, we compare how successful the attack is when applied to a classifier which has been trained directly with the real data than when it is applied to $h_S$ i.e., a classifier trained with the AM stimuli in place of the real data. We use the same ID architecture used for the previous experiments, for $h_S$ and for $h_T$. For $h_{mi}$ we used a small 4-layer fully connected network with 30-50 neurons per layer and elu activations. We trained $h_S$ and $h_T$ for Apnea-ECG and A3 for 100 and 10 epochs respectively. For both datasets we did not use the whole dataset but a randomly drawn sample of size 1000 for Apnea-ECG (500 training and 500 test) and 15000 for A3 (7500 training and 7500 test). In both cases we used 66.6% of the dataset to train $h_{mi}$ and 33.3% to evaluate. The experiment was repeated five times.

The Apnea-ECG results, in terms of kappa are: $h_T$: 0.120, $h_S$: 0.044. For the A3 study: $h_T$: 0.231, $h_S$: 0.042. These results correspond to the success of $h_{mi}$ in identifying whether previously unseen data were or were not used to train the model which is under attack ($h_T$ or $h_S$ depending on the experiment). Though $h_S$ does not directly use the training data during its training, the training-test separation is the same for $h_S$ and $h_T$. From these results we observe that even though the attack was for both datasets relatively unsuccessful both for $h_T$ and $h_S$, it was much more successful for $h_T$ than for $h_S$, as expected.
5.3 Class Recognition of Stimuli

An interesting observation regarding the proposed approach is that there seems to be disagreement across different algorithms trained on the real data, for the classes of the stimuli. This occurs even for the $X \rightarrow Y$ task, for which the stimuli maximize $h_T$’s output. We experimented with a linear SVM, a DNN (relu), a Random Forest (RF) ensemble with 50 trees, a Convolutional architecture identical to $h_T$ and three humans. We trained the four algorithms on the real data and evaluated them on a batch of 10000 from the MNIST stimuli dataset. For the humans, we randomly extracted a smaller batch of 100 stimuli and they performed digit classification on the batch. We used another 100 MNIST real test datapoints as baseline. The results are shown in Table 3. The examined methods and humans are not able to sufficiently identify the AM stimuli. As before, we attribute this to the inductive bias mismatch between the methods or humans and $h_T$. This is also supported by the much better recognition for the convolutional architecture identical to $h_T$. Notice also the strong correlation with Section 4.6. The models that had more difficulty to learn the patterns and generalize successfully to the real data are the ones performing the worst also in this experiment, as expected.

| Stimuli Class Recognition MNIST ($h_T$ trained for 2344 batches, Acc%) |
|-----------------------------|---------|---------|---------|---------|---------|
| Method                      | SVM     | RF      | DNN     | Humans  | Conv ID |
| Baseline                    | 93.91   | 96.69   | 97.69   | 97.00   | 99.20   |
| AM Batch                    | 26.78   | 28.85   | 38.84   | 17.30   | 79.39   |

Table 3: Class recognition of the AM stimuli from different algorithms trained on the real data and three humans. It is measured as accuracy over the stimuli batch.

6 Discussion

In this section we analyze important observations which occurred throughout our experiments.

6.1 Training Times

An interesting point of discussion is the required training time of the proposed approach. AM is by definition efficient, since assuming that we can easily reach the specified threshold, the AM generator only needs several steps to converge and satisfy the predefined condition. The total time accumulates linearly with the number of stimuli. In this work, we did not focus on training time, and we experimented mainly with larger numbers of stimuli. However, we can still obtain satisfactory performance even with a relatively small number of stimuli, e.g., with 3000 stimuli we can achieve an accuracy of 92% on MNIST. We needed close to 6 min to generate 3000 stimuli on a Nvidia-RTX 2000 Graphics Card.
Based on this discussion the overall training time depends on how many stimuli we need, which in turn depends on our performance requirements.

Additionally, the generalization capability of $G_{AM}$ can be leveraged to generate multiple stimuli after the AM generator has converged. However the stimuli from each run are very similar. Thus, additional diversity regularization is required. We leave this option for future work.

### 6.2 Performance Per Stimulus

We observe that the performance per stimulus is not the same among $h_T$ trained for different numbers of epochs. Figure 7 shows for MNIST the performance of $h_T$ trained for different numbers of epochs on the orange line. The blue line depicts how the $h_S$ with identical architecture performs when trained with 1000 data stimuli generated from the respective $h_T$. We observe that when $h_T$ is trained for a larger number of batches, the performance of $h_S$ for 1000 stimuli significantly drops. The best performance of $h_S$ in this experiment is obtained only for 390-790 batches of training. However, for 390-790 batches the performance of $h_T$ is generally low, and so the maximum reachable performance for $h_S$ is lower than for the $h_T$ that are trained for more iterations. For this reason, we choose the $h_T$ trained for 3125 batches for our MNIST experiments. This choice yields a good $h_T$ performance and not the worst AM dataset performance for 1000 stimuli. We use similar criteria for the training of $h_T$ for the apnea datasets.

### 7 Related Work

We first discuss related works which regard transferring knowledge between different classifiers. Then we turn our focus on anonymization for generative models. Finally, we analyze additional works that are useful and provide insights towards our approach.

#### 7.1 Knowledge Transfer

Recently, many new techniques for transferring knowledge have been proposed, especially with the goal of reducing the size of a DNN to decrease the execution
time and reduce memory consumption. Existing model compression techniques, e.g., via pruning or parameter sharing \cite{33, 34, 35, 36} can be considered as a form of knowledge transfer from a trained teacher to a student. Other types of methods perform knowledge transfer between different tasks and domains \cite{37, 38}. Furthermore, techniques exist that transfer knowledge from smaller DNNs to equal sized or larger DNNs to make them learn faster \cite{39} or to perform better than the original network \cite{40}. In the knowledge distillation method \cite{41}, the student network is trained to match the classes of the original data together with a modified version of the softmax output probabilities from the trained teacher network. This allows to control the steepness of the output class probability distribution. Romero et al. \cite{42} introduce fitnets, an extension of knowledge distillation to train thinner deeper networks (student) from wider shallower ones (teacher). Bucilúa et al. \cite{43} investigate the compression of large ensembles (like RF, bagged decision trees, etc.) via the use of a very small artificial neural network (ANN). As a universal approximator, the ANN is able to generalize to mimic the learned function of the ensemble. To train the ANN they create a larger synthetic dataset based on the real dataset that is labeled by the ensemble. Luo et al. \cite{44} use knowledge distillation on a selection of informative neurons of top hidden layers to train the student network. The selection is done by minimizing an energy function that penalizes high correlation and low discriminativeness. In \cite{45}, Papernot et al. use a similar student-teacher(s) scenario in order to train a student in a differentially private manner. The teacher ensemble is not released to the user, and also training of the student is done with real data (that are not in the private dataset) and a GAN used in a semi-supervised manner based on \cite{27}.

7.2 Generative Models and Anonymity

Regarding anonymization for generative models, many approaches exist which incorporate the differential privacy framework into the GAN framework \cite{9, 10, 46}. Contrary to these approaches, we focus on "hiding" certain data features. We achieve this by learning to avoid representations that leak these features, based on the output of an approximator of the true conditional. Thus, the proposed approach is data dependent and not a universal method incorporated in a mechanism. Other approaches manipulate the GAN framework in order to generate anonymized data (mainly images) for medical or general purposes \cite{47, 48}. Several approaches \cite{49, 20} utilize data-centric anonymization strategies similar to ours in order to hide sensitive information from potential adversaries. However, both of these aforementioned works modify or encode existing data-points with the use of adversarial frameworks. In contrast, we generate stimuli without directly accessing the real data. An interesting observation is that the vast majority of recent works utilizes modifications of the GAN framework to achieve their respective goals. The GAN framework is a natural choice in cases where data realism is important. As we do not have such a requirement we are interested in exploring an alternative approach.
7.3 AM and Other Works

To better understand the internal representations of a DNN, the AM technique is introduced in [13] for qualitative evaluation of higher-level internal representations of two unsupervised deep architectures. In [17], Nguyen et al. use a deep generative network to synthesize images that maximize the output of a neuron of a certain layer of the network. We base our approach on some of these insights.

For the purpose of clarity, a distinction should be made between our approach and approaches which include the use of trained convolutional filters. Convolutional filters are good feature extractors [50, 51] and these features can be used to train models [52], or be transferred from the feature space of a given domain to another domain [53]. However, we focus on training (from scratch) a new student classifier so that it can generalize well to a sub-region of the feature space defined from the AM (maximization or minimization). To do this, we take advantage of the same inductive bias which occurs by the use of a similar student architecture to that of the teacher. We do not use teacher layers as feature extractors. We use synthetic data sampled from the marginal distribution after AM is performed and not real data from another domain to train the student.

8 Conclusion and Future Work

The primary motivation for the proposed approach is to address the problem of limited training data availability due to privacy and sharing regulations. Our aim is to enable users to successfully train and customize models while minimizing the risk of identification for individuals who have contributed in the formation of the original dataset. As such, arbitrary users can benefit from private medical data sets to train or develop their own classification models. To achieve this we utilize AM in a generative manner, and create a multi-faceted dataset of stimuli that captures implicitly the class separation of the real data.

In this work we emphasize on a medical setting, and apply the proposed approach on the problem of sleep apnea detection. We utilize data from real-world clinical studies, and showcase its viability for the task. Furthermore, we evaluate on the task of digit recognition, and verify that the proposed approach is generalizable to different tasks and domains. Training with synthetic stimuli can yield promising results that are comparable or superior to a well-established generative method which can successfully produce realistic data. In this paper, we mainly evaluate on smaller classifiers potentially for use in a resource constrained environment. We experimentally show that we can utilize synthetic stimuli in place of real data to anonymize individuals that have contributed to the real dataset with their data. Furthermore, we utilize the proposed approach to produce a classifier that is more resilient against specific information leakage attacks, namely MI attacks.

In our ongoing and future work we address the customization of a student classifier $h_S$ towards the personal and unlabeled data of the end-user. In other
words, we aim to use NE for domain adaptation with only $h_T$ and the unlabeled data of the end-user as input, and a personalized $h_S$ as output. For both of these steps, we want to investigate the performance of NE if the classifier has differentially private guarantees.

Acknowledgement: This work was performed as part of the CESAR project (nr. 250239/O70) funded by The Research Council of Norway.

References

[1] CESAR. [https://www.mn.uio.no/ifi/english/research/projects/cesar/](https://www.mn.uio.no/ifi/english/research/projects/cesar/) [Online; accessed 5-6-2020].

[2] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[3] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuraman Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3–es, 2007.

[4] Cynthia Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

[5] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318, 2016.

[6] Bargav Jayaraman and David Evans. Evaluating differentially private machine learning in practice. In *28th {USENIX} Security Symposium ({USENIX} Security 19)*, pages 1895–1912, 2019.

[7] Yuheng Zhang, Ruoxi Jia, Hengzhi Pei, Wenxiao Wang, Bo Li, and Dawn Song. The secret revealer: Generative model-inversion attacks against deep neural networks. *arXiv preprint arXiv:1911.07135*, 2019.

[8] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. Deep models under the gan: information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–618, 2017.

[9] James Jordon, Jinsung Yoon, and Mihaela van der Schaar. Pate-gan: Generating synthetic data with differential privacy guarantees. 2018.

[10] Liyang Xie, Kaixiang Lin, Shu Wang, Fei Wang, and Jiayu Zhou. Differentially private generative adversarial network. *arXiv preprint arXiv:1802.06739*, 2018.
[11] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2015.

[12] Carlos R Ponce, Will Xiao, Peter F Schade, Till S Hartmann, Gabriel Kreiman, and Margaret S Livingstone. Evolving images for visual neurons using a deep generative network reveals coding principles and neuronal preferences. Cell, 177(4):999–1009, 2019.

[13] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. University of Montreal, 1341(3):1, 2009.

[14] Arthur S Reber. Implicit learning and tacit knowledge. Journal of experimental psychology: General, 118(3):219, 1989.

[15] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In Doina Precup and Yee Whye Teh, editors, Proceedings of the 34th International Conference on Machine Learning, volume 70 of Proceedings of Machine Learning Research, pages 214–223, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.

[16] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. arXiv preprint arXiv:1703.09387, 2017.

[17] Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In Advances in Neural Information Processing Systems, pages 3387–3395, 2016.

[18] Aravindh Mahendran and Andrea Vedaldi. Visualizing deep convolutional neural networks using natural pre-images. International Journal of Computer Vision, 120(3):233–255, 2016.

[19] Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. arXiv preprint arXiv:1602.03616, 2016.

[20] Clément Feutry, Pablo Piantanida, Yoshua Bengio, and Pierre Duhamel. Learning anonymized representations with adversarial neural networks. arXiv preprint arXiv:1802.09386, 2018.

[21] Aasm sleep apnea. https://www.medscape.com/answers/295807-53441/what-is-the-aasm-definition-of-apnea [Online; accessed 10-10-2019].

[22] Naresh M Punjabi. The epidemiology of adult obstructive sleep apnea. Proceedings of the American Thoracic Society, 5(2):136–143, 2008.
[23] Physionet, apnea-ecg. [https://physionet.org/physiobank/database/apnea-ecg/](https://physionet.org/physiobank/database/apnea-ecg/) 1999. [Online; accessed 10-10-2019].

[24] Stein Kristiansen, Mari Sunsteby Hugaas, Vera Goebel, Thomas Plagemann, Konstantinos Nikolaidis, and Knut Liestol. Data mining for patient friendly apnea detection. *IEEE Access*, 6:74598–74615, 2018.

[25] Gunn Marit Traaen, Lars Aakerøy, et al. Treatment of sleep apnea in patients with paroxysmal atrial fibrillation: Design and rationale of a randomized controlled trial. *Scandinavian Cardiovascular Journal*, (52:6,pp. 372-377):1–20, January 2019.

[26] Nox medical. (2019, october). nox t3 sleep monitor. [https://noxmedical.com/products/nox-t3-sleep-monitor/](https://noxmedical.com/products/nox-t3-sleep-monitor/) [Online; accessed 2-11-2019].

[27] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pages 5767–5777, 2017.

[28] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

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

[30] Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

[31] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

[32] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In 2017 *IEEE Symposium on Security and Privacy (SP)*, pages 3–18. IEEE, 2017.

[33] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[34] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

[35] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015.
[36] Suraj Srinivas and R Venkatesh Babu. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*, 2015.

[37] Junho Yim, Donggyu Joo, Jihoon Bae, and Junmo Kim. A gift from knowledge distillation: Fast optimization, network minimization and transfer learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7130–7138. IEEE, 2017.

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

[39] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[40] Tommaso Furlanello, Zachary C Lipton, Michael Tschannen, Laurent Itti, and Anima Anandkumar. Born again neural networks. *arXiv preprint arXiv:1805.04770*, 2018.

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

[42] Adriana Romero, Nicolas Ballas, Samira Ebrahimi Kahou, Antoine Chassang, Carlo Gatta, and Yoshua Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014.

[43] Cristian Bucilu, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541. ACM, 2006.

[44] Ping Luo, Zhenyao Zhu, Ziwei Liu, Xiaogang Wang, Xiaoou Tang, et al. Face model compression by distilling knowledge from neurons. In *AAAI*, pages 3560–3566, 2016.

[45] Nicolas Papernot, Martín Abadi, Ulfar Erlingsson, Ian Goodfellow, and Kunal Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[46] Chugui Xu, Ju Ren, Deyu Zhang, Yaoxue Zhang, Zhan Qin, and Kui Ren. Ganobfuscator: Mitigating information leakage under gan via differential privacy. *IEEE Transactions on Information Forensics and Security*, 14(9):2358–2371, 2019.

[47] Hoo-Chang Shin, Neil A Tenenholz, Jameson K Rogers, Christopher G Schwarz, Matthew L Senjem, Jeffrey L Gunter, Katherine P Andriole, and Mark Michalski. Medical image synthesis for data augmentation and anonymization using generative adversarial networks. In *International workshop on simulation and synthesis in medical imaging*, pages 1–11. Springer, 2018.
Håkon Hukkeløs, Rudolf Mester, and Frank Lindseth. Deepprivacy: A generative adversarial network for face anonymization. In *International Symposium on Visual Computing*, pages 565–578. Springer, 2019.

Ho Bae, Dahuin Jung, and Sungroh Yoon. Anomigan: Generative adversarial networks for anonymizing private medical data. *arXiv preprint arXiv:1901.11313*, 2019.

Douwe Kiela and Léon Bottou. Learning image embeddings using convolutional neural networks for improved multi-modal semantics. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 36–45, 2014.

Dario Garcia-Gasulla, Ferran Parés, Armand Vilalta, Jonatan Moreno, Eduard Ayguadé, Jesús Labarta, Ulises Cortés, and Toyotaro Suzumura. On the behavior of convolutional nets for feature extraction. *Journal of Artificial Intelligence Research*, 61:563–592, 2018.

Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Michael Xie, Neal Jean, Marshall Burke, David Lobell, and Stefano Ermon. Transfer learning from deep features for remote sensing and poverty mapping. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.
Appendix A  Accuracy, Specificity, Sensitivity for Apnea Detection Experiments

Appendix A shows the Accuracy, Sensitivity, and Specificity for the Apnea detection experiments for the different $h_S$ architectures for the Apnea-ECG and A3 datasets.

|  | $h_T$ Accuracy |  |  |
|---|---|---|---|
|  | Baseline  | WGAN  | AM  |
| AE ID:  | 93.59±0.10  | 86.39±0.44  | 92.06±0.18  |
| AE S:  | 93.81±0.24  | 85.04±0.42  | 91.36±0.29  |
| AE VS:  | 92.91±0.13  | 86.39±0.47  | 90.92±0.20  |
| AE L:  | 92.42±0.12  | 86.65±0.30  | 90.93±0.29  |
| AE LL:  | 92.93±0.13  | 85.77±0.14  | 90.56±0.15  |
| A3 ID:  | 83.67±0.09  | 80.37±0.15  | 81.72±0.33  |
| A3 S:  | 83.79±0.10  | 80.03±0.27  | 81.69±0.28  |
| A3 VS:  | 83.16±0.02  | 79.69±0.30  | 80.18±0.51  |
| A3 L:  | 83.74±0.07  | 80.32±0.15  | 82.13±0.10  |
| A3 LL:  | 83.42±0.05  | 79.72±0.28  | 81.19±0.13  |

Table 4: Accuracy of $h_S$ for AE and A3 experiments for different $h_S$ sizes when training with data generated from our method and from a conditional WGAN of similar size.

|  | $h_T$ Specificity |  |  | $h_T$ Sensitivity |  |  |
|---|---|---|---|---|---|---|
|  | Baseline  | WGAN  | AM  | Baseline  | WGAN  | AM  |
| AE ID:  | 94.11±0.12  | 87.97±0.41  | 90.24±0.21  | 92.89±0.29  | 84.23±1.03  | 94.54±0.22  |
| AE S:  | 93.34±0.27  | 87.52±0.23  | 89.36±0.36  | 94.45±0.60  | 81.64±1.02  | 94.10±0.69  |
| AE VS:  | 92.71±0.26  | 84.99±0.55  | 88.85±0.30  | 93.32±0.35  | 88.30±1.82  | 93.76±0.29  |
| AE L:  | 93.28±0.20  | 87.14±0.27  | 89.23±0.27  | 91.24±0.25  | 84.49±1.01  | 93.24±0.50  |
| AE LL:  | 92.08±0.33  | 86.26±0.12  | 88.35±0.32  | 94.10±0.46  | 85.10±0.46  | 93.58±0.08  |
| A3 ID:  | 80.72±0.43  | 76.57±0.71  | 75.81±1.05  | 86.18±0.32  | 83.62±0.63  | 86.76±0.38  |
| A3 S:  | 80.39±1.05  | 75.67±1.33  | 76.13±0.11  | 86.69±0.39  | 83.75±0.95  | 86.44±0.48  |
| A3 VS:  | 80.39±1.05  | 74.93±0.64  | 76.57±3.01  | 85.54±0.87  | 83.75±0.88  | 83.26±2.03  |
| A3 L:  | 81.29±0.78  | 74.38±1.07  | 77.61±0.65  | 85.84±0.62  | 85.38±0.65  | 85.98±0.40  |
| A3 LL:  | 81.31±0.81  | 73.95±1.26  | 79.23±1.27  | 85.22±0.67  | 84.65±0.78  | 82.86±1.01  |

Table 5: Specificity and Sensitivity of $h_S$ for AE and A3 experiments.
Appendix B  Architectures and Hyperparameters for the Models Used

In Appendix B, we show the architectures and hyperparameter values for the different models used in the experiments. We present the architectures of (1) the trained networks containing knowledge for the data distribution (conditional or joint, i.e., teachers T), (2) of the students that learn implicitly from the teachers, and (3) of other networks that are used as intermediates (namely the AM generator and the WGAN discriminator). Tables 3-8 showcase these results.

For Apnea-ECG, we use a batch size of 100. For A3 and MNIST, we use batch size of 128. All of random noise input are sampled from a standard normal distribution. The learning rate used for the Apnea-ECG and A3 experiments is 0.0001, and 0.001 for the MNIST experiments. We omit biases in all tables for simplicity. In all cases they correspond to the size of the output for the layer (number of channels on Convolutional or Transpose convolutional Layers).

B.1 Teachers T

We present the architectures of the models used as T for the different experiments in Tables 6, 7, 8.

| Architectures used          | Classifier $h_T$ (ID Arch.) |
|-----------------------------|-----------------------------|
| WGAN Generator              |                             |
| fc,D,relu: 31×60            | fc,D,relu: 60×360           |
| fc,D,relu: 60×180           | fc,D,relu: 360×180          |
| fc,tanh: 180×360            | fc,relu: 180×64             |
| fc,λ	anh: 360×60           | fc,softmax: 64×2            |

Table 6: Teacher Architectures used for the Apnea-ECG experiments (fc: Fully connected, input×output. The activation function used is shown next to the layer type separated with comma. If used, we show dropout as D).

| Architectures used          | Classifier $h_T$ (ID Arch.) |
|-----------------------------|-----------------------------|
| WGAN Generator              |                             |
| fc,relu: (128+10)×1024      | Conv,relu,MP: 1×32×5×5     |
| fc,relu: 1024×2048          | Conv,relu,MP: 32×28×5×5    |
| ConvTr,relu: 128×3×3×32    | fc,relu,D: (28×7×7)×1024   |
| ConvTr,relu: 32×4×4×28     | fc,softmax: 1024×10        |
| ConvTr,sigmoid: 28×4×4×1   | -                           |

Table 7: Teacher Architectures used for the MNIST experiments (Conv/ConvTransp: input channels×output channels×filter, MP: Max Pooling, fc: Fully connected, input×output).
## Architectures used

| WGAN Generator | Classifier $h_T$ (ID Arch.) |
|----------------|-----------------------------|
| fc,D,relu $(60+1) \times 120$ | Conv,relu,MP: $1 \times 4 \times 1 \times 16$ |
| fc,D,relu: $120 \times 1500$ | Conv,relu,MP: $16 \times 4 \times 1 \times 32$ |
| ConvTr,relu: $100 \times 4 \times 1 \times 16$ | Conv,relu,MP: $32 \times 4 \times 1 \times 64$ |
| ConvTr,tanh: $16 \times 4 \times 1 \times 32$ | fc,D,relu: $(64 \times 8) \times 64$ |
| ConvTr,tanh: $32 \times 4 \times 1 \times 64$ | fc,D,relu: $64 \times 32$ |
| ConvTr,tanh: $64 \times 2 \times 1 \times 1$ | fc,softmax: $32 \times 2$ |

Table 8: Teacher Architectures used for the A3 experiments (Conv/ConvTransp: input channels×output channels×filter, MP: Max Pooling, fc: Fully connected, input×output).

### B.2 Students

We present the architectures of the models used as students for the different experiments in Tables 9,10,11.

| Student Architectures used Apnea-ECG |  |
|-------------------------------------|--|
| **Classifier $h_S$** | **S Arch.** | **VS Arch.** | **L Arch.** | **LL Arch.** |
| fc,D,relu: | $60 \times 180$ | $60 \times 90$ | $60 \times 360$ | $60 \times 360$ |
| fc,D,relu: | $180 \times 90$ | $90 \times 45$ | $360 \times 180$ | - |
| fc,relu: | $90 \times 32$ | $45 \times 16$ | - | - |
| fc,softmax: | $32 \times 2$ | $16 \times 2$ | $180 \times 2$ | $360 \times 2$ |

Table 9: Student Architectures used for the Apnea-ECG experiments (fc: Fully connected, input×output. The activation function used is shown next to the layer type separated with comma. If used, we show dropout as D: Dropout).

| Student Architectures used MNIST |  |
|----------------------------------|--|
| **Classifier $h_S$** | **S Arch.** | **VS Arch.** | **L Arch.** | **LL Arch.** |
| Conv,relu,MP: | $1 \times 16 \times 5 \times 5$ | $1 \times 16 \times 5 \times 5$ | $1 \times 32 \times 5 \times 5$ | $1 \times 32 \times 5 \times 5$ |
| Conv,relu,MP: | $32 \times 14 \times 5 \times 5$ | $16 \times 12 \times 5 \times 5$ | $32 \times 28 \times 5 \times 5$ | - |
| fc,relu,D: | $(14 \times 7 \times 7) \times 512$ | $(12 \times 7 \times 7) \times 32$ | - | - |
| fc,softmax: | $512 \times 10$ | $32 \times 10$ | $(28 \times 7 \times 7) \times 10$ | $(32 \times 7 \times 7) \times 10$ |

Table 10: Student Architectures used for the MNIST experiments (Conv/ConvTransp: input channels×output channels×filter, MP: Max Pooling, fc: Fully connected, input×output).

### B.3 Other Networks

In this section we describe the additional intermediate architectures used in our experiments (i.e., WGAN Discriminators and the AM Generators).
### Table 11: Student Architectures used for the A3 experiments

| Classifier/Arch. | S Arch. | VS Arch. | L Arch. | LL Arch. |
|------------------|---------|----------|---------|---------|
| Conv,relu,MP:    | 1×4×1×8 | 1×4×1×4  | 1×4×1×16| 1×4×1×16|
| Conv,relu,MP:    | 8×4×1×16| 4×4×1×8  | 16×4×1×32| 16×4×1×32|
| Conv,relu,MP:    | 16×4×1×32| 8×4×1×16 | 32×4×1×64| -       |
| fc,D,relu:       | (32×8)×32| (16×8)×16| -       | -       |
| (MP),fc,D,relu:  | 32×16   | 16×8     | (64×8)×32| (32×8)×32|
| fc,softmax:      | 16×2    | 8×2      | 32×2    | 32×2    |

(Conv/ConvTransp: input channels×output channels×filter, MP: Max Pooling, fc: Fully connected, input×output).

#### B.3.1 AM Generator

We experimented with a variety of configurations as AM generator for the different experiments:

- **Apnea-ECG**: We use a 7-layer fully connected (512, 256, 256, 180, 60) MLP with relu activations and dropout in the first two layers. The noise input vector has size 512.

- **MNIST**: We use a convolutional-fully connected deconvolutional architecture, with 2 convolutional layers (1×32×5×5, 32×64×5×5), 1 fully connected (7×7×16×588), and 3 deconvolutional layers (12×32×2×2, 32×32×4×4, 32×1×4×4). All activations are relu except for the last layer which is use sigmoid. the input noise vector has shape (28,28,1).

- **A3**: We use a fully connected-deconvolutional architecture with a 512 input noise vector, with 2 fully connected (MLP) layers (512,1000) and 5 deconvolutional layers (200×128×3×1, 128×128×4×1, 128×64\times 4×1, 64×16×5×1, 16×1×2×1).

#### B.3.2 WGAN Discriminator

We experimented with a variety of configurations as AM generator for the different experiments:

- **Apnea-ECG**: We use a fully connected MLP with 4 fully connected layers, and an input of 61 (a 60 dimensional feature vector +1 a one dimensional condition) (120, 180,30, 1), with relu activations and dropout in the first 2 layers.

- **MNIST**: We use a convolutional network with an input of (28,28,1), 3 convolutional layers (1×128×5×5, 128×64×5×5, 64×32×5×5) and 2 fully connected layers (128, 1). We use in all cases relu activations, and dropout in the second to last fully connected layer. We pass the condition in the first fully connected layer.
• **A3**: We use a convolutional network with an input vector of 60, 3 convolutional layers (1 × 64 × 4 × 1, 64 × 32 × 4 × 1, 32 × 16 × 4 × 1), and 3 fully connected layers (120, 30, 1). We use in all cases relu activations, and dropout in the 2 fully connected layers. We pass the condition in the first fully connected layer.

**B.3.3 Association through Generalization: Model**

In this experiment we use the following architecture:

- We use a convolutional network and we normalize and concatenate the 4 features to create an input dimensionality of 240 (i.e., 4 × 60). We use 3 convolutional layers (1 × 8 × 5 × 1, MaxPool1D, 8 × 16 × 5 × 1, MaxPool1D, 16 × 32 × 3 × 1, MaxPool1D), and 3 fully connected layers (30-32,480, Out). Out depends on the model. Out = 8 for the adversary. Out = 4 for \( h_{TU} \). Out = 2 for the apnea detection classifier. We use in all cases relu activations, and dropout in the 2 fully connected layers.

**Appendix C Theoretical Analysis**

In order to achieve good results with the proposed approach, we investigate why and how the proposed method works. The main insights of this analytical investigation are presented in this section.

**C.1 General Insights**

For our discussion, we first define the true risk \( R(h) \) of a hypothesis \( h \) as follows:

\[
R(h) = \int_X \int_Y L(h(x), y)p(x, y) dx dy = -\int_X \sum_{c_j} p(y = c_j | x) \log h_{c_j}(x)p(x) dx
\]

The equality holds when we are using cross-entropy as \( L \), i.e., a tractable surrogate of the 0-1 loss, and is straightforward to derive (see Appendix C.2). We define the empirical risk \( R_{emp}^N \) as follows:

\[
R_{emp}^N(h) = \mathbb{E}_{x,y \sim \hat{p}(x,y)}[L(h(x), y)] = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c} y_i^c \log h_c(x_i)
\]

where \( h_c \) is the output of class \( c \), and \( y_i^c \) is class \( c \)'s element of the one hot encoded vector for input \( x_i \), \( y_i = \text{onehot}(c_i) \), and \( \hat{p}(x,y) \) the empirical joint distribution. Assuming a classifier \( h_T \) performs conditional density estimation and is trained to minimize its loss (we assume cross-entropy) with the empirical distribution corresponding to the samples and sample labels. Further, we assume that for a given loss a generalization bound applies for the hypothesis space \( H_T \) of \( h_T \).
The bound decreases towards 0 as the sample size $N$ increases to infinity. We assume that with probability $1 - \delta$, $\forall h_T \in H_T$:

$$R(h_T) \leq R_{\text{emp}}^N(h_T) + C(N, \delta)$$ (3)

Based on Eq.(1) for $h^* = \arg\min_{h \in H} R(h)$, and $h^N = \arg\min_{h \in H} R_{\text{emp}}^N(h)$ as $N$ increases we have that: $R_{\text{emp}}^N(h^*) \to R(h^*)$ (as for all other $h \in H$). As a result, since $C$ decreasing with $N$, we have one sided uniform convergence of the empirical risk to the actual risk. Due to the key theorem of learning theory, this means that the empirical risk minimization is consistent, i.e., the minimum of empirical risk converges to the minimum of the true risk $h^N \to h^*$. Alternatively, we can reach a similar conclusion, i.e., $R(h^N) \to R(h^*)$ via Theorem 1 and Remark 4 from [?]. For this to hold we need to assume that the loss is bounded, $R$ and $R_{\text{emp}}^N$ are $L$-Lipschitz with respect to the $l_2$-norm, and that the hypothesis space is parameterized with bounded parameter vector with respect to the $l_2$-norm. As we investigate in Appendix C.4 these constraints are met since they are needed for the next step of our analysis. For $h_S$, we want to minimize the true risk:

$$R(h_S) = - \int_X \sum_c p(y = c|x) \log h_S^c(x) p(x) dx$$

$$= - \int_X \sum_c (h_T^c(x) + d^c(x)) \log h_S^c(x) p(x) dx$$

$$= - \int_X \sum_c (h_T^c(x)) \log h_S^c(x) p(x) dx - \int_X \sum_c (d^c(x)) \log h_S^c(x) p(x) dx$$

$$= R_1(h_S) + R_\Delta(h_S)$$ (4)

From Eq. 4 we have access to $h_T$, but not to $d(x)$ (i.e., the difference between the true $p(y|x)$ and its estimate using $h_T$) and to $p(x)$. We also do not use the real data (sampled from $p(x,y)$). Thus, we focus on minimizing $R_1$, assuming that $h_T$ has generalized well. This would imply that $d$ is more likely to be small in $X$ and thus that $R_\Delta$ is more likely to be small.

However, we cannot minimize directly the first term since we do not have access to $p(x)$, or a sample of $p(x)$. The cross-entropy loss is minimized when the input distributions are equal, so if $h_S$ has the same architecture as $h_T$, the first term is minimized (trivially) for $h_S = h_T$, i.e., by simply copying the parameters of $h_T$ to $h_S$.

When $h_S$ does not have the same architecture as $h_T$, we use another probability distribution $p_Z(x)$ instead of $p(x)$. $p_Z(x)$ should have the same support as $p(x)$, or a support which has a non-empty set intersection with the support of $p(x)$, depending on the domain we are interested in. Thus, we try to minimize a ”distorted” version of the true risk, that is ”distorted ” in the sense that $p(x)$
is replaced by \( p_Z(x) \) and \( p(y|x) \) is replaced by \( h_T \):

\[
R_Z(h_S) = -\int X \sum_c (h^*_T(x)) \log(h^*_S(x)) p_Z(x) dx
\]

However again this minimization cannot be done analytically. Therefore, for \( h_S \), we create a dataset by drawing arbitrarily many samples from \( p_Z \) and \( h_T \) and training \( h_S \) to minimize the resulting empirical risk

\[
R_{Z_{emp}}^N(h_S) = \mathbb{E}_{y \sim h_T, x \sim p_Z}[ -y \cdot \log h_S(x) ].
\]

Here, we assume we generate probabilistically the labels \( y_z \) from \( h_T \) in a one-hot form. It is also possible to optimize directly for \( h_T \) as function. If Eq. (1) holds for \( H_S \) (hypothesis space of \( h_S \)), then the minimum of \( R_{Z_{emp}}^N(h_S) \), \( h_S^* \), converges to the minimum \( h_S^* = \argmin_{h_S \in H_S} R_Z(h_S) \) (\( h_S^* \rightarrow h_T^* \) as \( N \rightarrow \infty \)). However, if the hypothesis space \( H_S \) cannot express \( h_T \) perfectly on \( X \) such that after the minimization \( \forall x \in X, h_S(x) = h_T(x) \), it is not necessarily the case that \( h_S^* = h_T^* \), where \( h_T^* = \argmin_{h_S \in H_S} R_1(h_S) \). This is an inherent problem of the proposed method. However, if we make some additional realistic assumptions about \( R_1 \) and \( R_Z \), we can show that as we increase the parameter space of the student \( \Theta_S \) to that of the teacher \( \Theta_T \), assuming \( \Theta_S \subset \Theta_T \), the upper bound of the expectation of \( R_1 \) across the different optimized teacher parameters decreases when we optimize \( R_Z \). This means that as we increase \( \Theta_S \) and thus \( H_S \) we can expect better \( R_1 \) minimization by minimizing \( R_Z \) on a worst case. The interested reader can refer to Appendix C.4 for details.

In summary, based on the above discussion, assuming a good bound, minimizing \( R_{Z_{emp}}^N \) minimizes \( R_Z \). This in turn minimizes \( R_1 \) assuming the parameter space of the student is large enough that we have a good \( R_1 \) expectation. Finally, if \( R_\Delta \) is small \( \forall h_S \in H_S \), minimizing \( R_1 \) minimizes \( R \), which is our goal. In our case, we use AM to generate data for which our conditional distribution approximator \( h_T \) is confident about. As a result, during learning \( h_S \) focuses on these sub-regions of \( X \). Depending on the confidence level that we choose, and on \( h_T \) and its training, these regions could contain important information which could be generalized for the classification of the real data.
C.2 Risk for Cross-Entropy

Since we have multi-label classification, and we use cross-entropy as loss, we have:

\[
R(h) = \int_X \int_Y L(h(x), y)p(x, y)dxdy = \int_X \sum_{c_i} L(h(x), y_j)p(y = c_j|x)p(x)dx \\
= -\int_X \sum_{c_i} (\sum_{c_i} y_j^i \log h^{c_i}(x)) p(y = c_j|x)p(x)dx \\
= -\int_X \sum_{c_i} p(y = c_j|x) \log h^{c_j}(x)p(x)dx
\]  

where \( y_j \) is the one-hot encoding vector of the class.

C.3 Proposition

**Proposition 1** For \( f(x), g(x): X \rightarrow \mathbb{R}^N \), where \( X \) is a compact set and \( f \) and \( g \) belonging in a family of continuous, bounded functions \( G \), and loss \( L \), also continuous with \( L_{\min} \leq L \leq L_{\max} \), and \( L_{\min} > 0 \) in \( X \), we assume \( \forall x \in X \), and a given \( f \), \( \min_{g \in G} L(f,g)(x) \iff g(x) = f(x) \). Also, we assume prior distributions \( P = \{ p_1, p_2...p_M \} \), with support in \( X \). Then all expectations \( E_{x \sim p_i}[L(f,g)], i = 1...M \), are minimized iff \( \forall x \in X, f(x) = g(x) \).

Proof: First we show that if all expectations \( E_{x \sim p_i}[L(f,g)], i = 1...M \) are minimized then, \( \forall x \in X, f(x) = g(x) \). We use proof-by-contradiction, i.e., we first formulate the opposite statement, then show that it leads to a contradiction. In the following we assume that \( L,f \) and \( g \) are continuous and bounded in \( X \), and \( X \) compact.

Statement: Given \( f \in G \exists g \in G \text{ and } \exists p_i \in P \text{ s.t } \forall g_2 \in G \text{, } E_{x \sim p_i}[L(f,g)] \leq E_{x \sim p_i}[L(f,g_2)], \text{ and } \exists p \in X \text{ s.t } f(x) \neq g(x) \). We know that \( \forall x \in X, L(f(x), f(x)) \leq L(f(x), g(x)) \) by definition. This means that \( \forall p_i \in P \), \( E_{x \sim p_i}[L(f,f)] < E_{x \sim p_i}[L(f,g)] \).

This contradicts the above statement implying that \( \forall g_2 \in G \text{, } E_{x \sim p_i}[L(f,g)] \leq E_{x \sim p_i}[L(f,g_2)], \text{ for } f(x) \neq g(x) \) and some \( p_i \in P \).

Then we show that if \( \forall x \in X, f(x) = g(x) \), then all expectations of \( p_i \in P \) are minimized. If \( \forall x \in X, f(x) = g(x) \), then \( \forall g_2 \in G \), for which \( \exists x \in X \text{ s.t. } g_2(x) \neq f(x) \) we have by definition that \( \forall x \in X L(f,f)(x) \leq L(f,g_2)(x) \). As a result, \( E_{x \sim p_i}[L(f,f)] < E_{x \sim p_i}[L(f,g_2)], i = 1...M \).

C.4 Decreasing Bound on \( R_1 \) when minimizing \( R_Z \)

We perform the analysis for the decreasing bound for the expectation of \( R_1 \) in this Section. Before that we discuss preliminary notations and assumptions.
C.4.1 Preliminaries, Notations and Assumptions

Notational Conventions:

- \( \theta \): Parameter vector for classifier \( h \). \( \theta_S, \theta_T \), correspond to parameter vectors for the student and the teacher classifier respectively. When \( \theta \) is used this means a \( \theta \in \Theta_T \), regardless of student or teacher (i.e., the expression applies for both, since \( \Theta_S \subseteq \Theta_T \). See below).

- \( \theta^* \): Parameter vector after optimization has been performed. \( \theta^*_T \), corresponds to the optimization for the teacher, after the optimization of \( R_{emp} \). \( \theta^*_S \) corresponds to the optimized parameters for the student, after optimization for \( R_Z \) has been performed. Dimensionality for the parameter vector (i.e., how many parameters we have): \( |\theta_T| = d \).

- \( \Theta \): The parameter space of classifier \( h \) (see below). We use \( \Theta_T \) for the parameter space of the teacher and \( \Theta_S \) for the parameter space of the student. For our analysis (student has the same architecture with less weights per layer than the teacher) \( \Theta_S \subseteq \Theta_T \). For \( \Theta_S \), we increase this space, using a specific procedure, so that we transition from smaller networks (or in fact, networks which are identical to the teacher but contain many deactivated neurons) to larger networks (with less deactivated neurons). Thus, to showcase this increasing procedure we add the index \( n \) on \( \Theta^n_S \), and for \( n1 < n2 \), we have that \( \Theta^n_1 \subseteq \Theta^n_2 \).

- \( h \): The classifier architecture used. \( h_T \) refers to the teacher classifier, using \( \theta_T \) and \( h_S \) refers to the student classifier, using \( \theta_S \). When only \( h \) is used without index, we imply that the expression holds for both classifiers \( h_T \) or \( h_S \) is used. Also, we define as \( h^* \) the optimized \( h(\theta^*, x) \).

- \( H \): The hypothesis space for classifier \( h \).

- \( L(h_T, h_S)(x) = L(h(\theta_T, x), h_S(\theta_S, x)) \): The loss between two functions in \( x \). We assume cross-entropy loss for our analysis. As such for a given \( x \in X \) \( L(h_S, h_T)(x) = -\sum_c h^*_T(\theta_T, x) \log(h^*_S(\theta_S, x)) \).

- \( L_{max}(\theta^*_T) \): Maximum loss with respect to \( \theta^*_T \).

- \( L_{max} \): Maximum loss overall (for all \( \theta^*_T \)).

- \( B_{mx} \): Maximum allowed weight value for the teacher ( \( \Theta_T \) bounded).

- \( B_S \): Maximum allowed weight for the student for weight \( i \) (\( i \in \{0, d\} \)). Note: Always \( B_S \in [0, B_{mx}] \). One by one each \( B_S \) increases in value as we increase the parameter space of the student.

- \( R_1(\theta_T, \theta_S) = -\int_X \sum_c h^*_T(\theta_T, x) \log(h^*_S(\theta_S, x)) p_D(x) dx \).

- \( R_Z(\theta_T, \theta_S) = -\int_X \sum_c h^*_T(\theta_T, x) \log(h^*_S(\theta_S, x)) p_Z(x) dx \).
• $p_D(x), p_Z(x), p_{\theta_s}^{T_s}(\theta_T^*)$: Prior distributions. $p_D(x)$ corresponds to the real data marginal distribution, $p_Z(x)$ to the synthetic distribution that we use, and $p_{\theta_s}^{T_s}(\theta_T^*)$ to the teacher parameter distribution after the optimization of the teacher. Note that for this discussion, we assume that the support of $p_{\theta_s}(\theta_T^*), \text{supp}(p_{\theta_s}(\theta_T^*)) = \Theta_T^*$.

• $r$: Radius.

• $B(., r)$: For our analysis, $l_\infty$-hyper-ball with radius $r$ (so basically a multidimensional cube).

Regarding our assumptions, we firstly assume that all risks are continuously differentiable functions (and as a result $L$-Lipschitz), that the parameter space is bounded (in our analysis we bound the weights by the $l_\infty$-norm, and as a result the parameter space is also bounded by the encompassing $l_2$-norm), and that the loss $L$ is bounded (in $X$ given $\Theta_T, \Theta_S$ and the architectures of $h_T$ and $h_S$). Furthermore, we assume that both $R_1, R_Z$ are functions with a finite number of critical points or a finite number of regions of non-isolated critical points, and that for all activations $\phi, \phi(0) = 0$ (with the exception of the output where we assume softmax). Additionally, we assume that $X$ is compact.

C.4.2 Main Discussion

Our goal in this discussion is therefore to show that, we can bound the expectation of $R_1(\theta_T^*, \theta_S^*)$ assuming random variable $\theta_T^*$, and that this bound is decreasing as we increase the parameter space of $h_S, \Theta_S$ (with $\Theta_S \subseteq \Theta_T$). For this to hold we need to bound the loss $L$ (cross-entropy in our case), and thus we assume bounds on every element of the vector $\theta_T$, $-B_{ma} < \theta_T < B_{ma}, i \in \{0, d\}$ where $d$ is the total number of parameters of $h_T, |\theta_T| = d$. This together with the compactness of $X$, and the continuity of the activations of $h_T$ means that $L(h_T(\theta_T^*, x), h(\theta, x)) \leq L_{\max}(\theta_T^*) = \max_{x \in X, \theta \in \Theta_T} L(h_T(\theta_T^*, x), h(\theta, x))$. This holds since we assume softmax activations at the output of $h, h_T$. Further, we define $L_{\max} = \max_{\theta_T^* \in \Theta_T} \{L_{\max}(\theta_T^*)\}$.  

Now we focus on the construction of $\Theta_S$ and $H_S$. Architecturally, we assume that $h_S$ is identical to $h_T$. We assume that for any given parameter vector $\theta_S \in \Theta_S, |\theta_S| = d$, i.e., it has same dimensionality as $\theta_T$. However, we assume for the elements of $\theta_S$ that: $-B_{Si} < \theta_{Si} < B_{Si}, B_S \in [0, B_{ma}], i \in \{0, d\}$. So we allow connections to be deactivated (i.e., 0). If all input connections (including the bias) of a neuron are deactivated then a neuron is also deactivated (since for all activations $\phi \in h_T, h_S$ we have $\phi(0) = 0$). Thus, we can create any student whose architecture is thinner than the teacher. We cannot create students of smaller depth with this procedure since if all neurons on a certain depth are deactivated then regardless of input we will have a constant output (all classes equal $\forall x \in X$ since we use softmax output). Based on the above, we can construct a set $\{\Theta_S^n, n \in \mathbb{R}\}$, such that $\forall n_a, n_b, t, n_a < n_b, \Theta_S^{n_a} \subseteq \Theta_S^{n_b}$. We do this by starting with all weight bounds $B_{Si} = 0, i \in \{0, d\}$, and then picking a random connection $i$, increasing $B_{Si}$ from 0 to $B_{ma}$, then choosing a new
connection and repeating the process until $\Theta_S^3$ reaches $\Theta_T$. We use as index $n$ the total amount of "freed up" weight on $h_S$ (for example if we are on our third freed up connection and $B_{S_3} = q$, then $n = 2B_{mx} + q$)

By definition we know:

$$\forall x \in X, L(h_T(\theta^*_T, x), h(\theta, x)) = \min_i L(h^*_T, h)(x) \iff h^*_T(x) = h(x) \quad (6)$$

Note that $\theta^*_T$ may not be unique. For a $\Theta_S$ s.t $\theta^*_T$ does not exist in $\Theta_S$, but $\exists \theta^*_S \in \Theta_S$ such that $h_T(\theta^*_S, x) = h_T(\theta^*_T, x) \forall x \in X$, $R_1$ is also minimized for this $\theta^*_S$. This will not however affect our worst case discussion of the expectation of $R_1(\theta^*_T, \theta^*_S)$

$\theta^*_T$ follows a distribution with density $p_{\theta^*_T}(\theta^*_T)$ (potentially imposed from the initial weight distribution of $h_T$ together with potential inherent randomness during the optimization of $h_T$), and we assume that our minimization will always find the $\theta^*_S$ which minimizes $R_Z$ in $\Theta_S^3$. Then, $\theta^*_S$ is a function of $\theta^*_T$ such that:

$$\theta^*_S(\theta^*_T) = \begin{cases} 
\theta^*_T, & \theta^*_T \in \Theta_S^3 \\
\theta^*_n(\theta^*_T), & \theta^*_T \notin \Theta_S^3
\end{cases} \quad (7)$$

with $R_1(\theta^*_T, \theta^*_T) \leq R_1(\theta^*_T, \theta^*_S(\theta^*_T)) = \int_X L(h_T(\theta^*_T, x), h_S(\theta^*_S(\theta^*_T), x))p_D(x)dx$ due to the minimization of the loss. We continue assuming for simplicity that $\theta^*_T$ which creates the particular $h^*_T$ is unique $\footnote{If multiple $\theta^*_T \in \Theta_S^3$ exist for which $\forall x \in X, h_S(\theta^*_T, x) = h_T(\theta^*_T, x)$, then Eq. \ref{3} or Eq. \ref{5} would change so that for the $\theta^*_T \cup \theta^*_T \in \Theta_S^3 \ \theta^*_S$ takes the value of one of $\theta^*_T$ included in $\Theta_S^3$ it minimizes $R_Z(\theta^*_T, \theta^*_T) \forall \theta^*_T \in \Theta_S^3$. This means that we would get a tighter bound since the set $A = \{\theta^*_T | \theta^*_T \notin \Theta_S^3 \} \subset \Theta_S^3$, where $\Theta_S^3$ the subset of $\Theta_S$ of all $\theta^*_T$. Then $\forall \theta^*_T \in A$ we can use the bound $R_1(\theta^*_T, \theta^*_T) \leq R_{1MAX}$ instead of the $R_{1MAX}$.}$

For $R_{1MAX} = \max_{\theta^*_T \in \Theta_S^3} \{R_1(\theta^*_T, \theta^*_T)\}$, and $R_{1MAX}$ the maximum value for $R_1$, we have that $R_{1MAX} < R_{1MAX}$ (see section C.4). We also know that $\forall \theta^*_T \notin \Theta_S^3, R_1(\theta^*_T, \theta^*_n(\theta^*_T)) \leq R_{1MAX}$. Based on this we have:

$$E_{\theta^*_T \sim \theta^*_T} [R_1(\theta^*_T, \theta^*_S(\theta^*_T))] = \int_{\Theta_T} R_1(\theta^*_T, \theta^*_S(\theta^*_T))dP(\theta^*_T)$$

$$= \int_{\Theta_T} R_1(\theta^*_T, \theta^*_T)dP(\theta^*_T) + \int_{\Theta_T \cap \Theta_T} R_1(\theta^*_T, \theta^*_n(\theta^*_T))dP(\theta^*_T)$$

$$\leq \int_{\Theta_T} R_{1MAX} dP(\theta^*_T) + \int_{\Theta_T \cap \Theta_T} R_{1MAX} dP(\theta^*_T)$$

$$\leq R_{1MAX} \int_{\Theta_T} dP(\theta^*_T) + R_{1MAX} \int_{\Theta_T \cap \Theta_T} dP(\theta^*_T)$$

$$= R_{1MAX} P(\theta^*_T \in \Theta_S^3) + R_{1MAX} P(\theta^*_T \notin \Theta_S^3) \quad (8)$$

As we increase $\Theta_S^3$, and we know that $R_{1MAX} < R_{1MAX}$ the bound is decreasing, since more probability mass is put on the smaller $R_1(\theta^*_T, \theta^*_T)$ and less in the larger $R_{1MAX}$. However, this form is naive for 2 reasons: (1) It does
not decrease until we reach the last edge, and we have formed the full teacher network with the construction of $\Theta_S$. This is because when we add additional edges, we will not have any increase in the density $p_{\theta_T}^*$ which is inside $\Theta_S^*$ until we reach the final edge, since the increases for all previous edges are of lower dimensionality. (2) We have not utilized any property of the minimization of $R_Z$.

In order to investigate how the expectation changes as we increase $\Theta_S^*$ on a more useful setting that has actual utility, we make some additional observations.

For simplicity and since this is a worst case, continuing with the assumption that only the true optimized parameters of $h_T$, i.e., $\theta_T^*$ yields the function $h_T^*(x) = h_T(\theta_T^*, x) \forall x \in X$, in all of $\Theta_T$.

We then, define $d_{f_1} = \min_{\theta_T \in \Theta_T} \{\max_{\theta \in \Theta_T} \{R_1(\theta_T^*, \theta)\} - R_1(\theta_T^*, \theta_T^*)\} > 0$. We know that $d_{f_1} > 0$ from the analysis of Section C.4. Then, a fundamental observation for our analysis is that, due to continuity of $R_1$ and $R_Z$, and since $\theta_T^*$ is the strict global minimum in both $R_1$ and $R_Z$: $\forall \theta_T^* \in \Theta_T, \forall \delta_1 > 0$, we know that $\exists B(\theta_T^*, r_1(\theta_T^*))$, s.t. $\forall \theta'' \in B \cap \Theta_T, R_1(\theta_T^*, \theta'') \leq R_1(\theta_T^*, \theta_T^*) + \delta_1$. Similarly for $R_Z$ and $\delta_2$. Regarding the cases for which $\theta_T^*$ resides in the boundaries of $\Theta_T$, we use that part of the sphere that is inside $\Theta_T$.

Now we will use this property to construct a region close to $\theta_T^*$ so that, based on our assumptions if $\Theta_S^*$ has $\Theta_S$ sub-regions inside it, then the $R_1$ value of the student which is $R_Z$ optimized, is guaranteed to be small (i.e., smaller than $R_1$ plus an arbitrarily small $\delta_1$). We follow the following procedure:

- We choose $\delta_1 \in (0, d_{f_1})$.
- For a $\theta_T^* \in \Theta_T$, we choose the maximum radius $r_1(\theta_T^*)$ satisfying the above property for $\delta_1$, $r_{1\text{MAX}}(\theta_T^*) = \max_r \{r|\forall \theta'' \in B(\theta_T^*, r) \cap \Theta_T, R_1(\theta_T^*, \theta'') \leq R_1(\theta_T^*, \theta_T^*) + \delta_1\}$. Since in our analysis we want to use a single radius for which we want the property to apply $\forall \theta_T^* \in \Theta_T$, we choose $r_1 = \min_{\theta_T^*} \{r_{1\text{MAX}}(\theta_T^*)\}$.
- Then we transfer our analysis to $R_Z$. For a $\theta_T^* \in \Theta_T$, we define $R_{Z\text{OUTMIN}}(\theta_T^*) = \min_{\theta \in \Theta_T} \{R_Z(\theta_T^*, \theta)|\theta \notin B(\theta_T^*, r_1)\}$. We know that $\forall \theta_T^* \in \Theta_T, R_{Z\text{OUTMIN}}(\theta_T^*) > R_Z(\theta_T^*, \theta_T^*)$ since $R_Z(\theta_T^*, \theta_T^*)$ is strict global minimum. Based on this we define $d_{f_Z} = \min_{\theta_T^*} \{R_{Z\text{OUTMIN}}(\theta_T^*) - R_Z(\theta_T^*, \theta_T^*)\} > 0$.
- We define $\delta_Z \in (0, d_{f_Z})$. Using the same arguments that we used to create $r_1$, we construct $r_Z$.
- Then, because of the definition of $\delta_Z, r_Z$, we have that $\forall \theta_T^*, \text{ if } \exists \theta'' \in \Theta_S^*, \text{ s.t. } \theta'' \in B(\theta_T^*, r_Z), \theta_{S_Z} = \arg\min_{\theta \in \Theta_S} \{R_Z(\theta_T^*, \theta)\}, \text{ is inside } B(\theta_T^*, r_1)$. This means that:

\[
R_1(\theta_T^*, \theta_{S_Z}) \leq R_1(\theta_T^*, \theta_T^*) + \delta_1 \leq R_{1\text{MAX}} + \delta_1 < R_{1\text{MAX}} + d_{f_1} \leq R_{1\text{MAX}}
\]
Based on the above, as we increase our hypothesis space, we are searching for a hyper-ball with radius $r_Z$ and dimensionality equal to the number of parameters $\theta_T$ instead of a single point $\theta_T^*$.  

![Visualizations](image)

Figure 8: Visualizations. The first row shows for a small network, the "shell" network $h$, and the active regions that define the Teacher (blue) and the student (green). In the second row, we show the different cases in which $\Theta^n_S$ hits or misses the inner ball $B(\theta_T^*, r_Z)$. If $\Theta^n_S$ hits the inner ball, then the $\theta_T^*$ produced after the optimization of $R_Z$ will be inside $B(\theta_T^*, r_1)$ (outer ball), thus guaranteeing that $R_1(\theta_T^*, \theta_T^*_{S_E})$ is small. In the third row we show the equivalent problem when using the extended $\Theta^n_{S_E}$. Note that in all cases our spaces (Balls or parameter spaces) are hypercubes (or unions of hypercubes). In this illustration we show most of the spaces as circles for simplicity.

We observe that for a given $\Theta^n_S$ with dimensionality smaller than that of $\Theta_T$ (i.e., some weights do not contribute), if any point of a $B(\theta_T^*, r_Z)$, falls within $\Theta^n_S$, is equivalent with $\theta_T^*$ falling within an extended set $\Theta^n_{S_E} \supset \Theta^n_S$ with $\Theta^n_{S_E} = \{B(\theta, r_Z) | \theta \in \Theta^n_S\}$. However, based on the above discussion, when $\theta_T^* \in \Theta^n_{S_E}$, we have that the minimization of $R_Z(\theta_T^*, \theta) = \theta \in \Theta^n_S$, will find a good $\theta_T$ close to $\theta_T^*$, which is also good for minimizing $R_1$, and depending on the chosen $\delta_1$. Based on this, we use the following conventions for the 2 cases of

---

7Note here that if we do not assume uniqueness of minimization for $R_1, R_Z$, then if we have a finite number of point minimizers we can use similar arguments with the main discussion (use a union of balls close to the minimizers instead of a single ball).

If we have formations of minimizers, then in the case where we have only formations of the same dimensionality as the teacher (i.e., if we have plateaus) then by increasing the dimensionality of the student we increase the probability that $\Theta^n_S$ "hits" the minimizer structure without using hyperspheres. If we have lower dimensionality formations $F_B$ then for any of these we create the hyperspheres for any given point $\theta_T^*$ in the same way as in the main discussion, but with the additional exclusion of other points of the formation from the condition (where we have equal values for $R_1, R_Z$), defining similar properties for the spheres, and taking the union of the space which they create. In all these cases we obtain tighter bound.
\( \theta_{SZ}^* : \)

\[
\theta_{SZ}^* (\theta_T^*) = \begin{cases} 
\theta_T^*(\theta_T^*), & \theta_T^* \in \Theta_{SE}^n \\
\theta_n^*(\theta_T^*), & \theta_T^* \notin \Theta_{SE}^n 
\end{cases}
\]  

(9)

We know that, \( \forall \theta_1, \theta_2 \in \Theta_T, R_1(\theta_1, \theta_2) \leq R_{1MAX} \). Additionally, based on section C.4, and the above discussion we have that \( R_{1MAX}^* + \delta_1 < R_{1MAX} \). Then:

\[
E_{\theta_T^* \sim p_{\theta_T^*}}[R_1(\theta_T^*, \theta_{SZ}^* (\theta_T^*))] = \int_{\Theta_T} R_1(\theta_T^*, \theta_{SZ}^* (\theta_T^*))dP(\theta_T^*) \\
= \int_{\Theta_{SE}^n} R_1(\theta_T^*, \theta_T^*(\theta_T^*))dP(\theta_T^*) + \int_{-\Theta_{SE}^n \cap \Theta_T} R_1(\theta_T^*, \theta_n^*(\theta_T^*))dP(\theta_T^*) \\
\leq \int_{\Theta_{SE}^n} (R_{1MAX}^* + \delta_1)dP(\theta_T^*) + \int_{-\Theta_{SE}^n \cap \Theta_T} R_{1MAX}dP(\theta_T^*) \\
\leq (R_{1MAX}^* + \delta_1) \int_{\Theta_{SE}^n} dP(\theta_T^*) + R_{1MAX} \int_{-\Theta_{SE}^n \cap \Theta_T} dP(\theta_T^*) \\
= (R_{1MAX}^* + \delta_1) P(\theta_T^* \in \Theta_{SE}^n) + R_{1MAX} P(\theta_T^* \notin \Theta_{SE}^n)
\]  

(10)

The most important difference from the previous bound is, that for this bound, the extended student space has the same dimensionality as the teacher, although the actual student has a smaller dimensionality (less active weights). This means that as we increase \( \Theta_{SE}^n \), even when it has a smaller dimensionality than \( \Theta_T \), the probability that the student finds a good parameterization for \( R_1 \) by minimizing \( R_Z \) actually increases, because the probability that \( \theta_T^* \in \Theta_{SE}^n \) increases, and thus the expectation of \( R_1 \) decreases. This means that this bound is actually usable as a worst case scenario for a smaller than the teacher student, and it shows that as we increase the dimensionality of the student, the expected risk decreases even though we use a different prior \( p_Z \). Additionally, for a large choice of \( \delta_1 \), the drop of expectation as we increase \( \Theta_{SE}^n \) becomes small. However, for large \( \delta_1 \), \( r_z \) is generally expected to be larger, and thus the decrease in the bound more efficient than for a small choice of \( \delta_1 \) (i.e., we can reach the minimum expectation for a smaller \( \Theta_{SE}^n \) than if we chose a small \( \delta_1 \)). Finally, if we assume there exist multiple \( \theta_B(\theta_T^*) \), for which \( h_T(\theta_T^*, x) = h_T(\theta_B(\theta_T^*), x), \forall x \in X \), we follow a similar procedure as before (see footnotes 1,2) and end up with a tighter bound. In Figure 8 we include a visualization of the different spaces.

C.5 Why \( R_{1MAX} > R_{1MAX}^* \)

We show the statement of the title in the following points:

(1) We assume that, \( \forall \theta_1 \in \Theta_T, \exists \theta_2 \in \Theta_T, \text{s.t. } \exists x \in X \) with \( h(\theta_1, x) \neq h(\theta_2, x) \) (i.e., we have at least 2 different functions in \( H_T \)).

(2) Since all functions in \( H_T \) are continuous, and given all the requirements

43
we have fulfilled \((X, \Theta_T \text{ compact, } L \text{ continuous and bounded } h \text{ continuous and bounded etc})\), we derive from (1) (given Proposition 1): \(\forall \theta_1 \in \Theta_T \max_{\theta \in \Theta_T} \{R_1(\theta_1, \theta)\} > R_1(\theta_1, \theta_1)\).

(3) We define \(R_{1_{\text{MAX}}}^* = \max_{\theta \in \Theta_T} \{R_1(\theta_1, \theta_1)\}\). We know it exists because of all the compact and bounded constraints. Given (2) we know that for \(\theta_{1_{\text{MAX}}} = \arg\max_{\theta_1 \in \Theta_T} \{R_1(\theta_1, \theta_1)\}, \max_{\theta \in \Theta_T} \{R_1(\theta_{1_{\text{MAX}}}, \theta)\} > R_1(\theta_{1_{\text{MAX}}}, \theta_{1_{\text{MAX}}}) = R_{1_{\text{MAX}}}^*\)

(4) We define \(R_{1_{\text{MAX}}} = \max_{\theta_1 \in \Theta_T} \{\max_{\theta \in \Theta_T} \{R_1(\theta_1, \theta)\}\} \geq \max_{\theta \in \Theta_T} \{R_1(\theta_{1_{\text{MAX}}}, \theta)\} > R_{1_{\text{MAX}}}^*\) from (3).

Appendix D Adding Randomness

In order to create a multi-faceted stimuli dataset, we would ideally need the AM distribution \((p_{AM})\) to have a uniform support in the regions which we have defined as important, based on our threshold of acceptance.

Figure 9: Editor Design. Instead of generating datapoints to perform AM, we edit randomly sampled data.

To achieve this we want (1) to capture a wide variety of initial positions from the feature space, and then perform AM (so ideally we would want \(\text{supp}(p_{AM, \text{train}}) = \text{supp}(p)\)) and (2) add variability to the AM process itself. To achieve these points we perform the following during the execution of our algorithm:

- **Reinitializations:** We use our AM Generator to perform AM towards one class, we save the stimuli after it successfully surpassed the threshold of acceptance, and then we choose randomly another class to perform AM. Sporadically, we reinitialize the AM Generator, as the process can lead to numerical instability without reinitializations.

- **Random Thresholds:** In order to capture homogeneously the regions
that are above a certain acceptance threshold ($T_{\text{min}}$) for the class probabilities of $h_T$, we use a randomly changing threshold with $T \in [T_{\text{min}}, 1]$.

- **Use of larger initial weights** in the AM Generator: As we use reinitializations, we want to capture a large region of the feature space from the initial positions. For this reason when we use the AM Generator, we use larger than usual standard deviations in the weight initializations. However, this can lead to numerical instability during the AM Maximization. To solve this problem, we propose a more elegant solution described in the next point.

- **Use of AM Editor instead of AM Generator**: Instead of generating a synthetic stimulus (Generator), we sample randomly a datapoint from a uniform distribution with support in the feature space. Based on a scaled tanh function we use a network to perform edits to this datapoint until we satisfy our threshold target. This way, we have $\text{supp}(p_{AM,n}) = X$, which is very close to what we want in point (1). In order for the generated samples to be within the bounds of $X$, we include a rescale operator before feeding the stimuli to $h_T$. We experimented with this design in the MNIST experiments. The results surpassed the results when using an AM Generator with the same architecture, for the same number of stimuli. In Figure 9 we show the proposed design.

### Appendix E  Closest Neighbors per Dataset

In this Appendix, we present the closest neighbors (l2-norm) between the real and AM datasets (Figures 10,11).

![Figure 10: Closest neighbors between the Real(Left) and AM stimuli (Right) datasets for MNIST](image)

Figure 10: Closest neighbors between the Real(Left) and AM stimuli (Right) datasets for MNIST
Figure 11: Closest neighbors between the Real (Left) and AM stimuli (Right) datasets for Apnea-ECG (first row) and A3 (second row) (mV). Since A3 is a real-world dataset, with minimal pre-processing, we have many “inactive” datapoints that correspond to cases with sensor misplacement, bad sensor quality etc. In this case, the closest neighbor from the A3 study corresponds to such a datapoint.