UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Minimizing The Use of Personal Information in a Given Legacy System Via Data Synthesis

Minimierung der Verwendung Persönlicher Informationen in Einem Gegebenen Legacy System Mithilfe von Datensynthese

Bachelorarbeit

im Rahmen des Studiengangs **IT-Sicherheit**
der Universität zu Lübeck

vorgelegt von
Michael Brückner

ausgegeben und betreut von
Prof. Dr. Esfandiar Mohammadi

Die Arbeit ist im Rahmen einer Tätigkeit bei der id-netsolutions GmbH entstanden.

Lübeck, den 27. Mai 2020

Abstract

Implementing privacy preserving measures and minimizing the use of personal information in third party and legacy systems is challenging . We present a method of data pre-processing that achieves contract-limited privacy while staying fully compatible to a given system. We do that by building a synthesizer that is trained to create a privacy preserving version of a given input while preserving its utility. We formalise the notion of contract-limited privacy in this context. A hypothesis regarding a possible improvement for such a synthesizer is disproven by showing that unbounded additional privacy leakage could be introduced. We build and evaluate a highly successful synthesizer for various image classification datasets and expand on its versatility by applying different modifiers. First, we train the synthesizer to retain multiple utility properties of an input. Then we train it to fully retain the utility vector. Last, we add noise during the synthesis to improve the privacy aspect.

Zusammenfassung

Die Implementierung privatsphäre-erhaltender Maßnahmen, sowie die Minimierung der Verwendung persönlicher Daten in Fremdprodukten und Altsystemen ist herausfordernd. Wir präsentieren eine Methode der Daten-Vorverarbeitung, die Contract-Limited Privacy erreicht und gleichzeitig vollständig kompatibel zu einem gegebenen System bleibt. Dies schaffen wir, indem wir einen Synthesizer bauen, der darauf trainiert ist, eine privatsphäre-erhaltende Version eines gegebenen Inputs zu erschaffen, und dabei die Nützlichkeit zu erhalten. Wir formalisieren den Begriff der Contract-Limited Privacy in diesem Kontext. Eine Hypothese in Bezug auf eine mögliche Verbesserung eines solchen Synthesizers wird widerlegt, indem wir zeigen, dass zusätzliche unbeschränkte Privacy Leakage eingeführt werden könnte. Wir bauen und evaluieren einen sehr erfoglreichen Synthesizer für verschiedene Bildklassifizierungs-Datensets und erweitern seine Vielseitigkeit durch verschiedene Modifikationen. Zuerst trainieren wir den Synthesizer darauf, mehr als eine Nutz-Eigenschaft des Inputs zu erhalten. Dann trainieren wir ihn darauf, den vollständigen Klassifizierungsvektor zu erhalten. Zuletzt fügen wir Rauschen während der Synthese hinzu, um die Privatsphäre zu stärken.

Erklärung

Ich versichere an Eides statt, die vorliegende Arbeit selbstständig und nur unter Benutzung der angegebenen Hilfsmittel angefertigt zu haben.

_____

Lübeck, 20. März 2018

Acknowledgements

First and foremost, I want to express my gratitude to my supervisor Prof. Dr. Esfandiar Mohammadi for presenting me the opportunity to take on this for me new and interesting topic, and for encouraging and guiding me throughout every step of this thesis.

I further want to thank the id-netsolutions GmbH as a whole for funding and providing me with the technical foundations for my research. Also in particular from the id-netsolutions, I want to thank Jan Pimanow for supervising my work, and Julia Bröhan for supporting and helping me through technical difficulties.

Contents

1 Introduction

1.1 Privacy and Legacy Systems

The general data protection regulation (GDPR) imposes strict requirements that existing systems often do not satisfy. The GDPR mandates that the user gives their informed consent to the purposes for which the personal data can be processed. Moreover, solely those pieces of personal data shall be processed that are necessary for satisfying these pre-agreed purposes [gdp04].

It can be expensive to rewrite legacy-code to comply with those regulations. For external software, the inner workings of a program might even be unknown. An alternative approach is to anonymize the personal data such that it does not reveal any more than the information that is necessary for the pre-agreed purposes.

Along these lines, there is a rich body of literature for anonymizing data with the goal to protect personal data [WAB+18][MZ17][DR+14][Swe02]. However, prior work suffers from three shortcomings: either it reveals more information than necessary for fulfilling the agreed-upon purposes, or the resulting anonymized data is far less useful than the original data, or the approaches are computationally infeasible. According to the GDPR, it would suffice to sanitize the data such that the sanitized data cannot be used for anything beyond satisfying the pre-agreed purposes.

In this work, the term *contract-limited privacy* is used to describe that requirement. Contract-limited privacy is fulfilled when a system does only process the data necessary to fulfil the pre-agreed purposed.

Prior work did initial steps to build a computationally feasible and secure system, but did not achieve full contract-limited privacy [Ste19].

1.2 The Central Question

In this work we answer the following question: Is it possible to attain contract-limited privacy through pre-processing of a given dataset only? Working within those limitations, we analyse the general viability of our approach, its versatility under different circumstances, further potential improvements and use cases and of course its limitations.

1.3 Our Contribution: Information-Retaining and Privacy-Focused Data Synthesis

Our method is based on the concept of information-retaining data synthesis. Instead of feeding the original data into a system for processing, we aim to synthesize data using only the necessary information. This *synthetic data* keeps the original format and retains the utility, thus it is usable within the given legacy system while also ensuring contract-limited privacy of the stored data. The requested privacy is trivially fulfilled as the synthesized data can not contain any more information beyond the utility vectors it was built from.

Our synthesizer is a neural network. As such it has to be trained once to *learn* how to generate input to the utility function that yields a certain wanted output. Once learned, it can be applied to new data points, building a privacy preserving version of the data point that can be safely stored. The original input is then theoretically not needed any more and could be destroyed. Our method is application bound and is no general data anonymization tool. Instead it is a synthesizer tailored and trained for a specific system and for pre-determined utility functions.

One hypothesis that we evaluated was that we could conceptually improve the accuracy while keeping the architecture for the synthesizer. While we generally consider the utility function a black box, we take into account the possibility that intermediate results of that function might be available. In case of the synthesizer, increasing the amount of information the synthesizer would have to work with beyond the likelihood vector given by the classifier could lead to improved accuracy. In detail, we examine the hypothesis that it is possible to output the second to last layer through a filter, thus increasing the size of the input depending on the architecture of the classifier. It was important that this would not lead to any additional privacy leakage, however. We show that it is not possible to output any previous layer of the classifier, instead of the final classification, without potentially introducing unbounded privacy leakage, even when we allow an additional filter that could reduce the information that's given out.

To demonstrate the overall viability of our approach, we build a synthesizer for an image classification problem and show that we can achieve highly accurate results. We use different well known image datasets and several methods of evaluation to expand beyond only image classification. We also implement further modifiers and show that more than one utility property can be encoded, and that not only the class, but also the full utility vector can be retained.

1.4 The Structure of This Thesis

In chapter 2 we look into existing research with similar goals and define the problem and the precise scope of this thesis. The preliminaries 3 introduce all necessary concepts and terms needed to understand the methods and the results, and mention the frameworks that we used throughout the research to test and evaluate our ideas. In the methodology section 4 we describe the solution we implemented in greater detail on a higher level. The theoretical approach 5 presents our hypothesis of improvement of the synthesizer and proofs why that it not a viable approach in regards to privacy. The next chapter "Experiments" 6 then focuses on the implementation itself and its performance under different circumstances. We then evaluate the results in chapter 7 and interpret them in section with regards to the research question. Afterwards we present the limitations of our approach and our findings and discuss potential future areas of research in chapter 8. Finally, we conclude the work in section 9.

2 Problem Statement

In this chapter we first take a look into the work that's been done in this field so far and related topics that we pick up throughout the thesis. Next, we give a detailed outline of the problem that we research.

2.1 Related Work

With the rise of machine learning and cloud storage came the ability to process larger and larger amounts of data. This also gave rise to growing privacy concerns regarding this data. Research has since been conducted that aims to assure certain levels of privacy within the context of machine learning models [WAB+18]. However, most of these results focus on *building* privacy preserving systems [MZ17]. Furthermore, attacks against known privacy achieving systems are developed, for example against K-anonymity [NS08]. Additionally, other systems and algorithms, like SmallDB [DR+14], are relatively resource or time intensive in their computation. Another approach that's using data synthesis is built around training a new classification function on the synthetic data [BSG17].

We instead loosen the privacy requirements and achieve contract-limited privacy. We expand on previous work [Ste19] and synthesize new data based only on the utility of a given input. We do not aim to build a general data anonymization tool, but instead a application specific synthesizer. The advantages of this approach are two fold: Firstly, it does not need any adaption of a given system, but instead stays fully compatible with legacy or third-party software by being trained specifically on those systems. And second, it is secure against attackers with any amount of background knowledge, as none more than that is ever used at any point of the synthesis, and nothing more than the synthetic data is stored in the end.

The improvement in versatility especially with legacy systems offsets the relaxed privacy.

2.2 Problem Description

Legacy systems can often be too costly to adjust or retrain, or they might not even be accessible to a user, as would be the case if they're hosted off-premise by a third-party for

example. A user might still want to achieve privacy and thus needs a way to store and process their data without losing the accessibility of the legacy network.

Ideally, to achieve contract limited privacy, any information that is not output by the utility function and thus is not needed for a set goal, should be removed from storage. The most simple solution would be to just store the utility output. However, for specific reasons outside a user's control, it might not be feasible to use the utility output in further processing steps directly, as they might expect inputs of the original shape.

Given such a black box utility step, how do we synthesize new data points based only on useful information, keeping the utility while staying fully compatible to the black box? We build a synthesizer that uses the black box's output to generate a new synthetic input that gives the same utility result.

Consider the following situation: There exists a third-party or legacy system that cannot be adapted. This system takes as input the original data points, extracts certain features from it and outputs those features in some form. Since the storage of the data does not fulfil contract-limited privacy as more than the needed features are saved, that system has to be adapted in some form. However, as the legacy system is considered a black box to us, we only have the ability to alter the inputs. The trivial solution of storing only the utility results is considered not possible due to technical reasons within that black box. We now aim to devise a data pre-processing step that allows us to fulfil contract-limited privacy while staying compatible to the given black box system.

3 Preliminaries

In this chapter the important terminology and concepts are defined and explained.

3.1 General Data Protection Regulation

Ever since the introduction of the General Data Protection Regulation 2016/679 (GDPR) privacy became a focus point for companies and customers alike. It regulates the use, protection and the transfer of personal information in the EU. Its implementation poses many with challenges not only in the usage, but also the storage of data in production systems. Coupled with the rise of cloud storage, the correct handling of private information in compliance with the GDPR is a complex task. And while new systems can since be built with privacy in mind from their conception, legacy systems often present a special problem in that regard. Completely rebuilding or retraining a legacy system to comply to the GPDR might be costly, and in case of third-party software potentially impossible.

3.2 Artificial Neural Networks

In this work often simply referred to as neural network, artificial neural networks describe a computational approach that's inspired by its biological equivalent. An (artificial) neural network consists of a series of *layers*, each of which is made up of *neurons*. Between the *input layer* and the *output layer*, any number of so called *hidden layers* apply certain functions to the output of the previous layer. The specific form and function of each layer depends on the type of layer chosen, and how its parameters are set.

The weights of such a network are a defining feature, allowing it to *learn*. Certain types of layers include weights on the connections to the following layer, determining how much of an impact each connected neuron has on the output by multiplying it with its corresponding weight. Given a set of weights θ and a loss function L quantifying how close the result of the neural network N is to the expected result, the goal of training is to find weights to reduce the loss

$$\theta^* := argmin_\theta L(N_\theta(X), Y)$$

for a data-set D of tuples of the input X and the expected output Y

$$D = (X_i, Y_i) \text{ for } X = (X_i)_i \text{ and } Y = (Y_i)_i$$

The weights are automatically updated during this learning phase. The exact workings of this function and most of the various kinds of layers are not necessary for the scope of this work though and will not be described further.

3.3 Fully Connected Layer

One notable layer, however, is the fully connected layer, also known as dense layer. As the name implies, in this layer each neuron is connected to all neurons of the following, with each connection having its own weight w_{ji} from neuron j to neuron i. This layer is commonly found in simple networks and is also usually the last layer of a classification network, reducing however many neurons down to the amount of different classes, each neuron representing a single class.

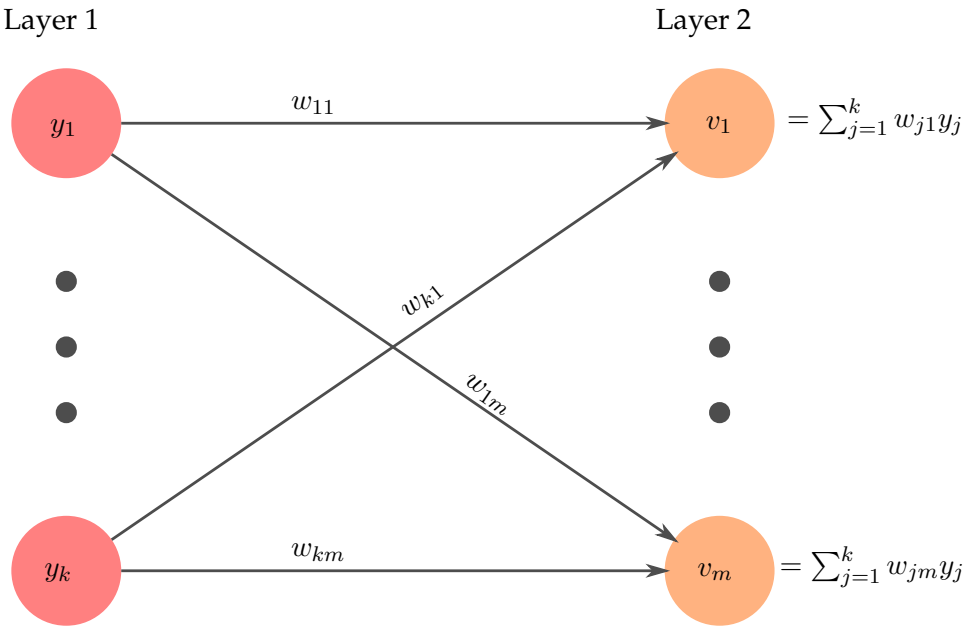

Figure 3.1: General Overview of a Dense Layer

3.4 Utility and Utility Functions

The utility of an input is defined through the useful features of the input within a given context. This could for example be the class of the input in an image classification problem. A utility function u maps an input $x \in \mathbb{R}^d$ to a utility value $u(x) \in \mathbb{U}$. The form of $\mathbb{U}$ depends on the type of problem that is discussed. Prevalent cases are multi-class and multi-label classification and regression.

In multi-class classification, each input belongs to exactly one of k classes. As such, $\mathbb{U} = y_1, ..., y_k$, with one entry for each class. Normally, the output describes a likelihood vector with the probabilities of the input belonging to each given class, so $\sum_{i=i}^{k} y_i = 1$. The result it the class with the highest probability $arg\max_i(y_1, ..., y_k)$.

Multi-label classification allows for the inputs to belong to more than one class, so $\mathbb{U} = y_1, ..., y_k, \forall y_i : y_i \in [0, 1]$, as each class has its own independent probability.

In regression problems, the goal is to find the overall relationship of the outcome to one or more input features. For example in linear regression, a straight line that best fits all given data points and can be used to predict new values based on the inputs.

3.5 One-Hot Encoding

One-hot encoding usually refers to the way that labels of a multi-class classification problem with k different classes are represented. One-hot ground-truth labels are a probability vector C with k values, one for each class. Since only one class can be correct, each value is 0 except for the one at the index of the correct class, which is 1.

3.6 Autoencoders

In general, autoencoders are a type of neural network used to find an efficient encoding of the input. Through a series of layers, the autoencoder reduces the dimension of the input, thus mapping it to a point in the code space. After this *encoding* step, the autoencoder tries to regenerate the original input during the *decoding*. The chosen loss function depends on the goal. Commonly used ones are categorical-crosssentropy and (root) mean square error. Figure 3.2 shows the basic components in an overview.

Definition 3.1 (Autoencoder). An autoencoder is a form of neural network consisting of two functions, an encoder $E : \mathbb{R}^i \Rightarrow \mathbb{R}^c$ followed by a decoder $D : \mathbb{R}^c \Rightarrow \mathbb{R}^i$. $\mathbb{R}^i$ describes the input space, $\mathbb{R}^c$ the code space.

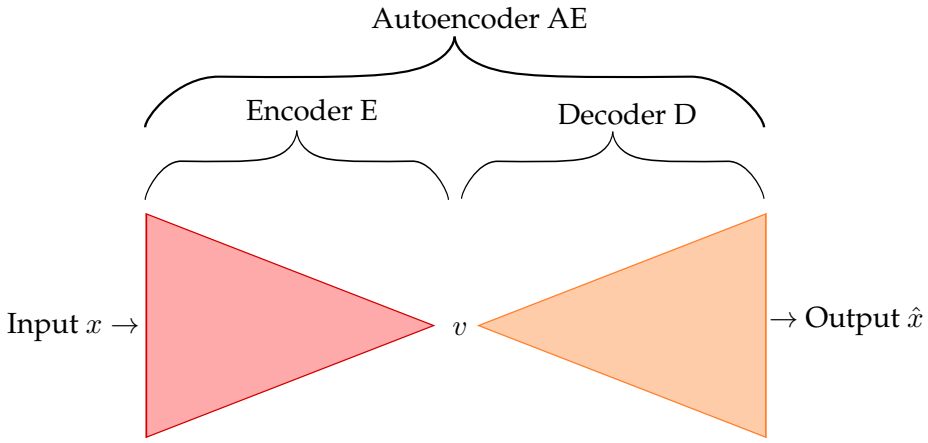

Figure 3.2: General Autoencoder

3.7 Data Synthesizer

In the context of this thesis, a data synthesizer is a trained neural network. It takes as input the original data and outputs new data in the same format based only on the utility of the original data. It retains the utility, which means that the utility function yields the same result when applied to the original and its corresponding synthetic input.

3.8 Contract-Limited Privacy

One main concept of this work is *contract-limited privacy*. The idea is based on Article 6(1) of the GDPR. In particular, we are interested in the point stating that "Processing shall be lawful only if and to the extent [...] processing is necessary for the performance of a contract to which the data subject is party". In this context, we use the term *contract-limited privacy* to describe exactly that. Contract-limited privacy is achieved if only that information is stored and used that is necessary for the fulfillment of the contractual obligations between two parties. Mathematically speaking, we define a synthesizer S to be contract-limited privacy preserving through the follow formula, given a utility function u:

$$\forall x, x' : \left\|u(x) - u(x')\right\| \leq \alpha \geq 0 \Rightarrow \left\|S(x) - S(x')\right\| \leq L \cdot \alpha$$

That describes that if the utility of two data points is close in euclidean distance in code space, then the synthesized versions of those inputs have to be close to each other as well.

3.9 Multi Function Utility

We describe as multi-function utility the property of an input to be used on more than one utility function, thus extracting different features independently. For example, one might want to extract the pose and the facial expression from an image x of a person using two different utility functions $pose(x)$ and $face(x)$.

That means, for a set of n utility functions u_i with $i = 1, ..., n$ and an input x: $\forall i : u_i(x)$ is valid as in a useful result.

3.10 Error and Distance

The error is a quantification of the difference between two likelihood vectors with k classes. Most important for this work is the *Mean Square Error*

$$mse = \left(\frac{1}{k}\right) \sum_{i=1}^{k} (target_i - pred_i)^2$$

and its root

$$rmse = \sqrt{mse}\,,$$

and the *Mean Absolute Error*

$$mae = \left(\frac{1}{n}\right) \sum_{i=1}^{n} |target_i - pred_i|\,.$$

The error can also act as distance quantification between two vectors, or as a loss metric for the neural network.

3.11 General Terms

A short overview of other terms that are used throughout this thesis.

1. *Ground-truth*: The true labels / classes for each input that were given with the dataset.
2. *Target*: The set of target labels that a model tries to reproduce. In most cases this is equivalent to the ground-truth.
3. *Likelihood vector*: The output of the classifier. A probability vector with one value for each class describing the likelihood that the input belongs to each class.

4. *Prediction*: The label that the classifier predicts for a given input, determined as the class with the highest probability in the likelihood vector.
5. *Accuracy*: A value describing the percentage of correct predictions in comparison to the target over a set of inputs.

3.12 Frameworks

The experimental part of this work was implemented using the open source Tensorflow library (GPU Version 2.0.0) [AAB+15] with Python in version 3.6. Further the Keras based high-level Tensorflow API was used. This allowed for a high degree of customizability alongside quick prototyping capabilities. The calculations were run on on Amazon Web Services (AWS) Sagemaker instance of type ml.p2.xlarge with GPU support.

4 Methodology

In this chapter, we describe the solution to answering the central question of this thesis. We then share initial thoughts that we had about the approach and how to improve it. Lastly, we give the exact details on the technical background that was used to implement the solution.

4.1 The Proposed Solution

As described in chapter 2, our goal is to remove all information from the original data that is not necessary to fulfil the contractual obligations, so that it can be stored safely, achieving and maintaining contract-limited privacy. Instead of editing the original data-points, however, we create new, synthesized versions per input based only on that necessary information, its utility. This additional step does not change the size nor the format of the original inputs, meaning that they stay fully compatible to a given legacy system. For that, we adapt the general concept of an autoencoder. The autoencoder is effectively fed with the output of the utility function and trained to generate new data in the input's format that yields the same utility.

Since we encode the utility-vector, this approach complies with the requirement for contract-limited privacy from prior work:

Lemma 4.1. *Let $\mathcal{T}$ be an encoder-decoder data synthesizer with Gaussian noise such that $\mathcal{T}(x) = D(E(x) + \eta)$, where $\eta \sim \mathcal{N}(0, I\sigma^2)$. If $\exists L > 0 \; \forall \; x_0, x_1 \in \mathbb{R}^d \; . \; |u(x_0) - u(x_1)| \leq \alpha \implies ||E(x_0) - E(x_1)|| \leq L\alpha$ then $\mathcal{T}$ satisfies (ϵ, δ)-contract-limited privacy for $\sigma \geq \frac{\sqrt{2ln(1.25/\delta)}L\alpha}{\epsilon}$ [Ste19].*

Since we have $E := u$, $|u(x_0) - u(x_1)| \leq \alpha \implies ||E(x_0) - E(x_1)|| \leq L\alpha$ is trivially fulfilled with $L = 1$. As a corollary, if d-times 1-dimensional Gaussian noise is used, we get, by standard composition theorems, ($d \cdot \epsilon$,$d \cdot \delta$)-contract-limited-privacy.

First in chapter 5, we look into the theoretical side of this approach. We present an idea on how to improve the synthesizer and show why it is not possible to implement that without potentially introducing considerable privacy leakage. After that, we develop and implement proofs of concept that show the viability of our synthesizer. Here we also

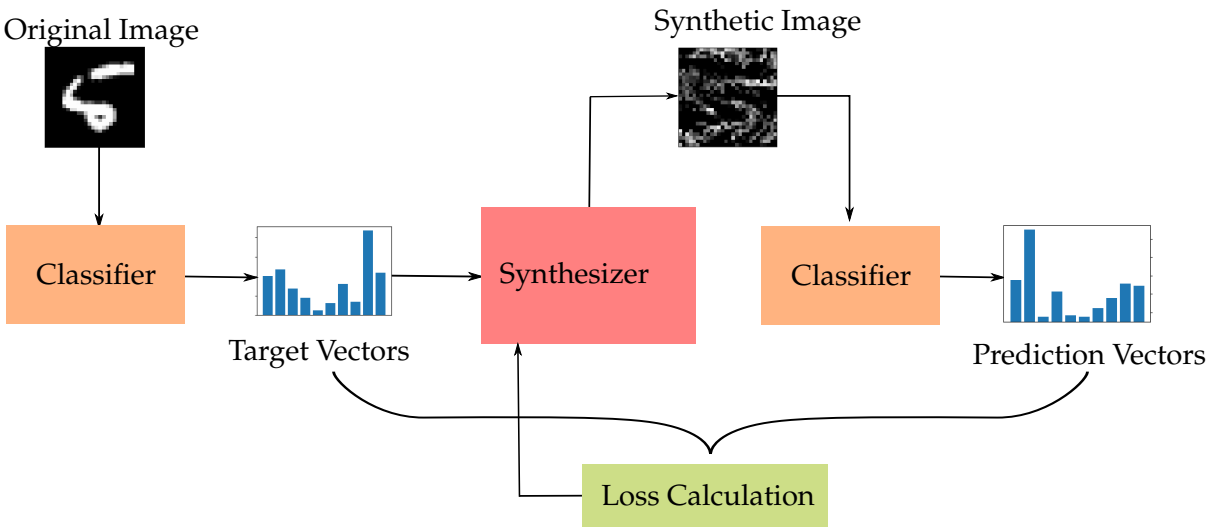

Figure 4.1: The Basic Process of the Synthesis.
(Note that in the implementation, the initial and ending classifiers are part of the synthesizer and are just separated here for clarity.)

showcase further ideas that can be applied to the base synthesizer. Either, we improve its privacy aspect through the introduction of noise during the synthesis or its versatility by showing that more than one utility function can be kept applicable to the synthetic data. Further we show that not only the class, but also the complete vector can be retained.

4.2 Multi-Function Utility

Often times in real life scenarios, inputs might have more than one class of features that can be extracted. For example, an image of a person could be classified by the expression on their face as well as their overall pose. This can be done by training different classifiers on the same images but with a different set of labels. We say that this input has *multi-function utility* 3.9.

When there are several different utility functions that can each be applied to any given input x, it would be possible for our data synthesizer $synth(x)$ to create that amount of images per input, each of which could then be used by their respective utility function. However, as that would significantly increase the needed storage space and potentially introduce further complexity in handling the data within a legacy system, it would be preferable to generate a single output image $\hat{x}$ that upholds the multi-function utility of the original input.

Definition 4.2 (Multi-Function Utility Property of Synthesized Data). Let u_i with $i = 1, ..., n$ be one of n utility functions that can be applied to an input x. Let $\hat{x} = synth(x)$ be the output of the synthesizer. The multi-function utility property of a data synthesizer is then described through

$$\forall x, i : u_i(x) = u_i(\hat{x})$$

To achieve this multi-function utility we effectively use multiple in- and outputs to train a a single synthesizer.

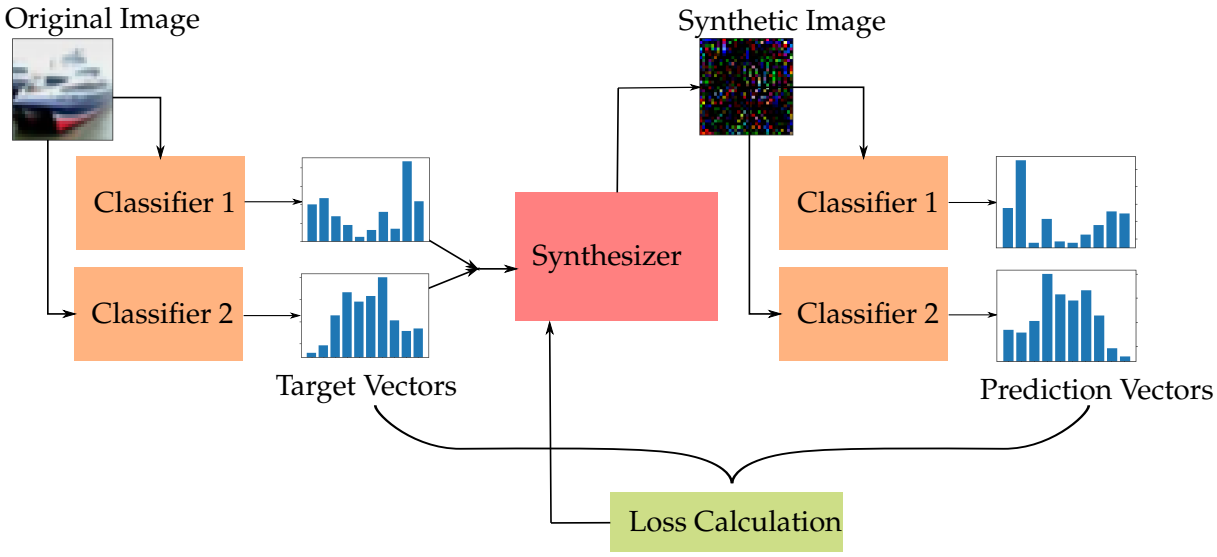

Figure 4.2: The Process during Multi Function Utility Training with two Classifiers

As figure 4.2 shows, the classifiers act independently from each other. Instead of having one target and prediction, we now have multiple. The input vectors are concatenated before going into the synthesizer. The loss is calculated for each classifier independently and then added up to give the total loss.

4.3 Implementation

We implemented all of the solutions with the Tensorflow 2.0.0 Framework [AAB+15] in Python 3.6. The experiments were run on a GPU enabled instance of Amazon Web Services of type ml.p3.2xlarge [aws], leading to highly performant tests with usually less than 30 seconds per training epoch. more details on the implementation can be found in the appendix 10 and the code.

5 Theoretical Approach

In this section we examine our theoretical hypothesis and through this gain a deeper understanding of the workings of our methods. One goal of this thesis was to evaluate whether it is possible to feed the second to last layer of the classifier through an additional filter to the synthesizer to improve the accuracy. In theory, more information to build the synthetic images from should lead to a better and more accurate synthesis. Outputting another layer than the last one of the classifier is a plausible idea to achieve that. It is important that this must not introduce unbounded additional privacy leakage, hence the use of a filter function to remove only as much additional information as necessary.

However, we show why that requirement is too strong for our approach and why it is most likely not possible to implement a useful filter when the last two layers of the utility function are dense layers.

5.1 Outputting Additional Information

In general, a neural network classifier aims to decrease the dimensions of the input gradually, extracting new features in each layer, until it reaches the output featuring only a single neuron per potential class. Our classifiers were build with two or more dense layers towards the end to achieve that dimensionality reduction. One might think it could be possible to use the second to last layer, filtering it and giving that information to the synthesizer. The idea is, that a filter can reduce the privacy leakage to an acceptable amount while still giving more information to the synthesizer.

5.2 Privacy Leakage and Lipschitz Continuity

To achieve privacy, we want our code space to be as small as possible while still keeping the utility of the inputs. It would trivially be possible to map all inputs to a single point in our code space, but that would leave no discernible information to synthesize a useful output that can be traced back to its input. Our mappings have to stay discernible, and to determine the privacy leakage on each layer, we have to introduce a new metric.

We can determine that the slope of a function within our synthesizer is directly correlated to its privacy leakage. The steeper the slope, the higher the privacy leakage. The

reasoning is that with a steep slope, small changes of the input lead to bigger changes in the output, thus spacing the outputs further out and making them more easily identifiable from one another.

To determine privacy leakage, we make use of the concept of Lipschitz continuity of real-valued functions and its inverse.

Definition 5.1 (Lipschitz Continuity of Real-Valued Functions). A function $f : \mathbb{R} \Rightarrow \mathbb{R}$ is Lipschitz continuous if there exists a real, positive constant L such that $\forall x_1, x_2 \in \mathbb{R} : \|f(x_1) - f(x_2) \leq L \|x_1 - x_2\|\|$

The idea of that property is that the maximal slope of the function is bounded by the Lipschitz constant. Thus, the smaller the Lipschitz constant, the smaller the privacy leakage of that function. We now proof that a preceding dense layer can always have a potentially higher privacy leakage even when most filter functions are applied. Through this, we show that it is not generally possible to remove the last layer of the utility function without introducing privacy leakage when those layers are fully connected while keeping a certain utility.

Technically, the last layer is usually followed by a softmax function defined as

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} .$$

This function is not Lipschitz continuous, but is needed for the loss calculation of categorical crossentropy. We avoided that problem by applying softmax only during the loss calculation and leaving the actual results linear. The exact details of that process are not needed, as the hypothesis fails with or without softmax at the end.

5.3 Counter Proof to the Initial Hypothesis

This proof consists of three parts. The first two lemmas help us deepen our understanding of Lipschitz Continuity within this context, the last lemma shows the unbounded privacy leakage when this approach is applied.

Lemma 5.2. *Let u be the utility function. If $u = E \circ g$ and for some function F*

$$\forall x, x' : \left\|u(x) - u(x')\right\| \leq \alpha \wedge$$
$$\forall y, y' : \frac{\|g(y) - g(y')\|}{\|y - y'\|} \geq \frac{1}{L} \frac{\|F(y) - F(y')\|}{\|y - y'\|}$$

implies

$$\forall x, x' : \left\|(E \circ F)(x) - (E \circ F)(x')\right\| \leq L \cdot \alpha \, .$$

Further, as a special case, if F is L' Lipschitz continuous,

$$\begin{aligned} &\forall x, x' : \left\|u(x) - u(x')\right\| \leq \alpha \wedge \\ &\forall y, y' : \frac{\|g(y) - g(y')\|}{\|y - y'\|} \geq \frac{L'}{L} \end{aligned}$$

implies

$$\forall x, x' : \left\|(E \circ F)(x) - (E \circ F)(x')\right\| \leq L \cdot \alpha \, .$$

Proof. We assume $u = E \circ g$ and for some function F

$$\forall x, x' : \left\|u(x) - u(x')\right\| \leq \alpha \wedge \tag{5.1}$$

$$\forall y, y' : \frac{\|g(y) - g(y')\|}{\|y - y'\|} \geq \frac{1}{L} \frac{\|F(y) - F(y')\|}{\|y - y'\|} \, . \tag{5.2}$$

We will show that

$$\forall x, x' : \left\|(E \circ F)(x) - (E \circ F)(x')\right\| \leq L \cdot \alpha \, .$$

Due to assumption (5.1), it is sufficient to show that

$$\frac{\|(E \circ F)(x) - (E \circ F)(x')\|}{\|u(x) - u(x')\|} \leq L \, .$$

We define $u := E \circ g$ and $y_x := E(x)$.
It suffices to show

$$\frac{\|F(y_x) - F(y_{x'}))\|}{\|g(y_x) - g(y_{x'})\|} \leq L \, .$$

Hence, we get

$$\begin{aligned}
&\frac{\|F(y_x) - F(y_{x'}))\|}{\|g(y_x) - g(y_{x'})\|} \cdot \frac{\|y_x - y_{x'}\|}{\|y_x - y_{x'}\|} && \leq L \\
\Leftrightarrow &\frac{\|y_x - y_{x'}\|}{\|g(y_x) - g(y_{x'})\|} \cdot \frac{\|F(y_x) - F(y_{x'}))\|}{\|y_x - y_{x'}\|} && \leq L \\
\Leftrightarrow & && \frac{\|g(y_x) - g(y_{x'})\|}{\|y_x - y_{x'}\|} \geq \frac{1}{L} \frac{\|F(y_x) - F(y_{x'}))\|}{\|y_x - y_{x'}\|}
\end{aligned}$$

which holds by assumption (5.2).

If F is L′ Lipschitz continuous

$$\frac{\|F(y_x) - F(y_{x'})\|}{y_x - y_{x'}} \leq L'$$

Then it suffices to prove

$$\frac{\|g(y_x) - g(y_{x'})\|}{\|y_x - y_{x'}\|} \geq \frac{L'}{L} \,. \qquad \square$$

Lemma 5.3. *If $u = u' \circ h$ and*

$$\forall y, y' : L'' \left\|y - y'\right\| \leq \left\|h(y) - h(y')\right\| \leq L' \left\|y - y'\right\|$$

$$\forall x, x' : \left\|E'(x) - E'(x')\right\| \leq L \left\|u'(x) - u'(x')\right\|$$

then

$$\forall x, x' : \left\|(E' \circ h)(x) - (E' \circ h)(x')\right\| \leq L \frac{L''}{L'} \left\|u(x) - u(x')\right\| \,.$$

Proof.

$$\begin{aligned}
&\forall x, x' : \left\|h(E'(x)) - h(E'(x'))\right\| && \leq L \frac{L''}{L'} \cdot \underbrace{\left\|h(u'(x)) - h(u'(x'))\right\|}_{\leq L' \|u'(x) - u'(x')\|} \\
\Leftrightarrow &\underbrace{\forall x, x' : \left\|h(E'(x)) - h(E'(x'))\right\|}_{\geq L'' \|E'(x) - E'(x')\|} && \leq LL'' \cdot \left\|u'(x) - u'(x')\right\| \\
\Leftrightarrow &\forall x, x' : \left\|E'(x) - E'(x')\right\| && \leq L \left\|u'(x) - u'(x')\right\|
\end{aligned}$$

which holds by assumption. $\square$

These two lemmas state that if the last layer h is Lipschitz and inverse Lipschitz continuous, the layers in the middle can be removed as long they are Lipschitz continuous as well.

Lemma 5.4. *Let y be the vector of values y_j with $j = 1, ..., k$ of a generic dense layer 1 within a neural network, then let v be the vector of values v_i with $i = 1, ..., m$ on the following layer 2. Let w_{ji} be the weight from the j-th node on layer 1 to the i-th node on layer 2, and g be the linear transformation between two layers with $g(y) = v$ and $v_i = \sum_{j=1}^{k} w_{ji} y_j$. Further let F be a filter function, and $\tilde{y} = F(y)$*

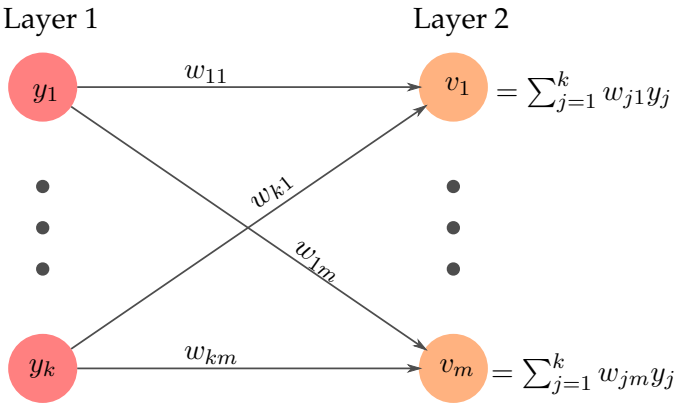

Figure 5.1: For direct reference: Fully Connected Layer 3.3

We show that

$$L||g(y) - g(y')||_2 \not\geq ||F(y) - F(y')||_2.$$

Proof. Proof by Contradiction
We assume

$$\begin{aligned} & L||g(y) - g(y')||_2 && \geq ||F(y) - F(y')||_2 \\ \Leftrightarrow \quad & L\left|\sqrt{\sum_{i=1}^{m}(v_i - v_i')^2}\right| && \geq \left|\sqrt{\sum_{j=1}^{k}(\tilde{y}_j - \tilde{y}_j')^2}\right| \\ \Leftrightarrow \quad & \left|\sqrt{L^2 \sum_{i=1}^{m}(\sum_{j=1}^{k} w_{ji} y_j - \sum_{j=1}^{k} w_{ji} y_j')^2}\right| && \geq \left|\sqrt{\sum_{j=1}^{k}(\tilde{y}_j - \tilde{y}_j')^2}\right| . \end{aligned}$$

With the monotony of the square root function, it would suffice to show that the inequality holds for each addend of the i-term.

$$\forall i : (\sum_{j=1}^{k} L w_{ji}(y_j - y_j'))^2 \geq \sum_{j=1}^{k}(\tilde{y}_j - \tilde{y}_j')^2 . \qquad \lightning \tag{5.3}$$

Irrespective of the filter function F we apply or L we choose, this proof does not hold.

It is obvious that the right side of the inequality is monotonically increasing

$$\sum_{j=1}^{k} \underbrace{(\tilde{y}_j - \tilde{y}'_j)}_{\geq 0}^2 \geq 0 \,.$$

Assuming $y \neq y'$ we can even say

$$\sum_{j=1}^{k} (\tilde{y}_j - \tilde{y}'_j)^2 > 0 \,.$$

For the left side, however, we cannot make such an assumption. As it has to hold for any $y, \tilde{y}$, this means that there could be cases where $Lw_{ji}(y_j - y'_j))^2 < 0$ even when we apply weight regularization to keep all weights positive. □

5.4 Conclusion on that Approach

Our approach to completely remove the last layer of the classifier and giving the synthesizer more information that way within our system did not work as intended. As we could not find another way to introduce additional information in a privacy-retaining way, the rest of this thesis focuses on working solely with the utility vector. The experiments show, however, that this is enough to achieve useful results.

6 Experiments

The following chapter presents the experimental base of this thesis. First we discuss why the implementation offers a scientifically sound basis for meaningful results in the context of a proof of concept. Then the input datasets used for the experiments are presented. Next, the different experiments and setups are shown and lastly our methods for evaluating the different cases are presented.

6.1 The Basis of the Experiments

The goal of the experiments is to show that the aforementioned pre-processing step produces useful synthetic data that upholds a certain level of anonymity. We assume that the overall legacy system extracts certain pieces of information from the input, and outputs its useful information, its utility, in some format. That utility function can be a black box, as no details about its inner workings are explicitly needed.

As the range of possible utility functions is vast, we decided to use multi-class (single-label) 3.4 image classification as exemplary function. That classifier is in itself a neural network and as such can be complex enough to represent further different functions. At the same time, image classification is easy to visualize and offers well known and studied datasets that we use in the experiments. The classifier is trained separately, and acts as black boxes towards the synthesizer. Depending on the training dataset, we can further examine different levels of complexity. The details of their architecture are only important insofar as they are shown to be sufficiently deep and complex. We also take their accuracy into account during evaluation.

We decided to build and train the classifiers ourselves to be able to control the complexity if needed. Further this would have helped to implement the hypothetical idea 5 for improving the synthesizer. In the final versions of the synthesizers, however, no additional background knowledge was used at any point. The classifiers act as total black boxes and as such in full accordance to our initially defined limitations.

The choice of image classification came on one hand from its ease of visualization and on the other hand its relative ease of implementation. As we wanted to build a proof of concept, it was important that we could rapidly build and especially train our synthesizer with reasonable computational needs. With our different methods for evaluation, the

results can be indicatively used to go beyond image classification. Of course, image classification tasks are also prevalent in many real life applications and represent a potential field of operation.

For this work we aimed for standard architectures for our models in both the classifiers and the synthesizer to keep the results as generally applicable as possible. The specific model architecture can be adapted to fit the needs of a productive environment. We further tried to keep the synthesizers as simple as possible to show that highly accurate synthesizers do not necessarily need to be deep models.

6.2 Input Datasets

In this work, we focus on inputs in the form of images. Three well known datasets are used: the MNIST dataset of handwritten digits [LCB10] and the two datasets CIFAR-10 and CIFAR-100 [KH+09].

The MNIST dataset consists of overall 70.000 grayscale images of size 28x28 pixels and their respective labels. Each image features a single, centered, handwritten digit. A predefined subset of 10.000 images is used for evaluation, the remaining 60.000 for training.

The CIFAR-10 dataset consists of 60.000 colour images with 3 colour channels in the scale of 32x32 pixels over 10 mutually exclusive classes, each class having 6000 images. Again, 10.000 images are for testing, the remaining 50.000 images for training.

The CIFAR-100 dataset is analog to CIFAR-10, except that it has 600 images per class over 100 classes.

Further details of each of those sets are not needed for this work.

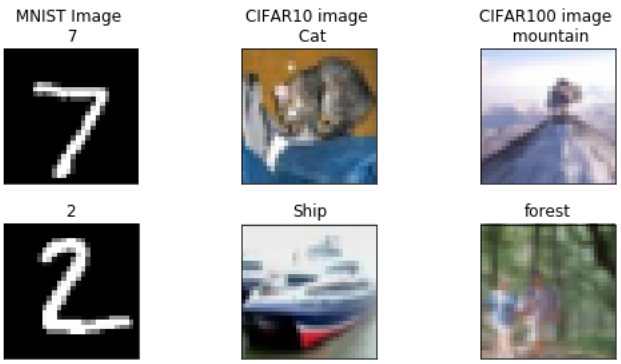

Figure 6.1: Example images from each dataset with their respective label.

6.3 General Setup

The basis for all the experiments consists of two parts: A classifier and a synthesizer. The classifiers are trained once for each dataset using rather simple convolutional architectures that reach acceptable levels of accuracy 10.1. To keep the black box idea in mind, the classifiers are not adapted in any particular way to the task at hand. The synthesizers allow for a broader range of customisability. We tested several base architectures and then adjusted them depending on the goal of each experiment.

There were three cases based on the availability of ground-truth labels and the goal, and additional variants that could be applied to each case [1].

1. In the first case, the training set has the original ground-truth labels for the inputs available. This might, for example, be the set on which the classifier was trained or externally verified outputs from the classifier itself. Using these labels to train the synthesizer leads to a synthesizer focused solely on the reconstruction of the correct class - the rest of the likelihood vector is not taken into consideration. Under most circumstances, this is an easier case. Theoretically, the synthesizer could output one representative per class and simply map the inputs to their respective representative.

   The loss function for our neural network is *categorical crossentropy*, a usual loss function for this use case. The ground-truth is *one-hot encoded* 3.5. The details of categorical crossentropy are not needed and will be omitted at this part.
2. Next, we take the classifier's output as chosen ground-truth. The difference in operation for the synthesizer is small, except that it has no mismatch between the predicted input class and the target class. The synthesizer solely learns to reproduce the input utility vector. This case is akin to using a perfect classifier in the first use case.
3. The last case changes the goal: Instead of learning to get the correct label or class, the synthesizer now has to match the complete likelihood vector. In this case, the synthesizer has to retain all of the information that is input, which means that simple representatives per class are not sufficient any more. This is the hardest and most interesting case. Success here shows the viability of the synthesizer for multi-label classification problems, and high potential for regression problems. Since we now want to recreate a vector, we use the mean-square-error 3.11 as loss function. That way, we minimize the overall difference between the synthetic and the target vector.

In addition to these cases in their basic form, the following variants can be applied:

1. Upholding the multi function utility of the inputs. The synthesizer does not only

[1] The corresponding code adjustments are generally straight forward, like fitting the input and output shapes to the dataset and choosing the ground-truth for the network during training.

train to reconstruct the output of a single classifier, but of multiple classifiers at once that all work with the same input. This is interesting as it represents a probable real life use case.

As non of our datasets have independent features labelled, we use the CIFAR-10 and CIFAR-100 classifiers as two functions and use the CIFAR-100 dataset as input. Obviously, this mismatched set does not lead to useful results with the CIFAR-10 classifier. Instead we refer to the second case above and aim for the retention of the predicted nonsense labels.

2. Adding Gaussian noise during synthesis. After the synthesizer has been completely trained, we can add Gaussian noise with mean zero and varying standard deviations on the input's likelihood vector and analyse the behaviour of the synthesizer on the perturbed input data. The noise is applied to each value of the likelihood independently, meaning we apply 1-dimensional Gaussian noise per class.

   By perturbing the code space like that, we improve the privacy of the resulting synthetic dataset. When we add enough noise such that the noisy likelihood vector could belong to a different input, an attacker cannot clearly determine the original input any more.

6.4 Approach for Evaluation

In general, we determine the accuracy of our synthesizer by comparing the classification of the synthetic images with our choice of target data. This yields a percentage of correctly labelled synthetic images, the accuracy.

In the first case our target data is the ground-truth provided by the dataset. In this approach, we expect the accuracy to be roughly bounded by the accuracy of the classifier itself. Given a rather accurate classifier, the synthesizer will mostly try to reproduce the predicted vector; unless there is a consistent error in the classifications. In case the classifier consistently misjudges a specific type of input, for example a '7' as a '1' in the MNIST dataset. It could be possible for the synthesizer to fix that flaw by learning to produce a synthetic image that classifies as '7' whenever the classifier outputs a likelihood vector that shows ambiguity between '7' and '1'. However, for that to work, the error of the classifier would have to be consistent and appear often enough in the dataset for the synthesizer to learn that feature.

Both the MNIST and the CIFAR-10 dataset share the number of classes. Thus, the synthesizer gets the same amount of input and the central difference is potentially the complexity of the classifier and the shape of the synthetic image. To further analyse the capa-

bility of the synthesizer, we use the 100 classes of the CIFAR-100 dataset.

Due to the relatively small amount of images per class, a comparatively simple architecture for the classifier is generally not accurate enough. We use this, however, to determine how our synthesizer fares against more complex models. With less accurate models, the classification function could be of a considerably higher degree. Additionally, there might be more ambiguity in the utility vector, which is even more challenging for the synthesizer. Nevertheless, the second use case from above will still show us how well the synthesizer can retain the inaccurate classifications.

The last case can only partially be evaluated with *accuracy*. The reconstruction of the likelihood vectors is represented with the mean-absolute-error between the synthetic and the target vector. This metric shows the mean absolute difference between the original and the synthetic vector. This allows us to compare the synthesis between the different datasets in particular.

For multi function utility, we use the same method as for the base model, except that the accuracy is assessed for each classifier separately.

The noisy synthesizer is most interesting in combination with the third case. When our goal is to only retain the label, the question would mostly be whether or not the noise was enough to flip the label to another one. To properly evaluate the use of noise, we have to further analyse the results of the classifier.

We are interested in three additional metrics: The *mean absolute distance* and the *root mean squared distance* of the likelihood vectors within each class, and the average difference of the likelihood of the most likely and the second most likely class. The first two distances give us an idea of the clusters of different classes in the code space. This helps us determine how much noise we need to add in total to a vector for it to be potentially attributed to another input from the same class. The last difference shows us on average how much noise we can add before the likelihood vector evaluates to another class.

7 Results

This chapter presents the results of the experiments. First, we explain a few terms that we use and then show the basic synthesizer architecture that has proven to be highly accurate. After that, we show the four different cases on different datasets: The basic synthesizer, the multi-function utility synthesizer, the likelihood retaining synthesizer and a noisy version of the likelihood retaining synthesizer. All cases show very promising and successful results.

7.1 Terms

The following tables use shortened column names that are described here.

1. "Classifier Acc.": Percentage of correct predictions compared to the ground-truth given the *original images*.
2. "Synth. against ground-truth (gt)": Percentage of correct predictions compared to the ground-truth given the *synthetic images*. This is the general accuracy that the synthetic dataset has with the respective classifier.
3. "Synth. against classifications": Percentage of correct predictions compared to the *predicted classes of the original images* given the synthetic images. This shows how well the synthetic images retain the original inputs' predicted classifications, similar to the situation with a perfectly accurate classifier.

7.2 The Model Architectures

Generally, each synthesizer is approximately analogue to the CIFAR-10 base model which operates as described in figure 4.1.

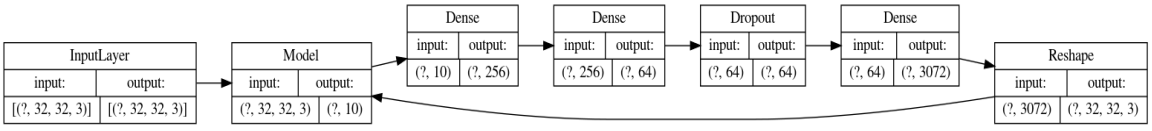

Figure 7.1: The General Model for the Synthesizers.

As depicted in figure 7.1, the input is fed into the classifier model (simply named

"Model" in the figure). The resulting classification then goes through two dense layers and a dropout layer with a dropout rate of $p = 0.4$. That layer is a regularization layer, dropping each connection with a probability of p. That way, over fitting can be reduced. Over fitting appears when a network is not generalised enough and instead only works reliably on the training data, and not on new, unseen inputs.

In the end, one more dense layer brings the dimension up to the needed amount ot pixels to form an image, which are then reshaped into the input format and once again classified for loss calculation.

The training happens on the utility vector from the final classification, while the interesting output comes from the intermediate reshape layer that represents the result in image form.

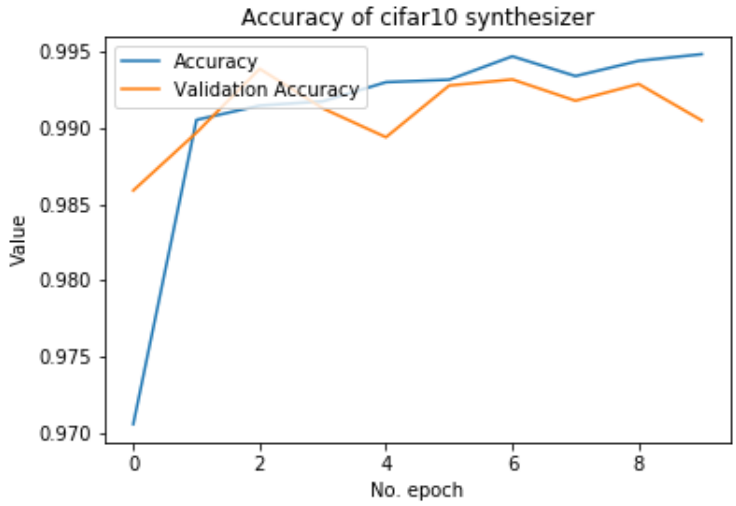

Figure 7.2: Example: The accuracy of the basic CIFAR-10 synthesizer through the epochs during training.

Figure 7.2 shows the accuracy during the training of the synthesizer on the CIFAR-10 dataset. We can see a rather quick convergence towards the final accuracy. With the simplicity of the model, that behaviour is expected. Each of our tested models showed similar behaviour, reaching the final accuracy within less than 5 epochs. As such we refrain from presenting all the individual training graphs here, a few more can be found in the appendix 10.3. Based on these results, we trained our synthesizers for five to ten epochs.

7.3 The Basic Synthesizer

Our first test was the basic synthesizer with the first use case. The goal is to generate a synthetic image that retains the information from the original utility vector upon classification. Our target data for now is the original ground-truth label with one-hot-encoding.

Table 7.1: Results for the Basic Synthesizer Without Additions Trained on Ground-Truth

Dataset	Classifier Acc.	Synth. against ground-truth	Synth. against classifications
MNIST	0.9929	0.9929	0.9962
CIFAR-10	0.8276	0.838	0.925
CIFAR-100	0.5613	0.5718	0.8197

In table 7.1 we see the results of the basic synthesizer. According to our expectations, the accuracy of the synthetic dataset is approximately bounded by the accuracy of the classifier itself. As the last column shows, the synthesizer mostly retains the input classifications with high accuracy. However, that accuracy sinks quickly with more complex models and inputs like CIFAR-100. The mismatch between the ground-truth labels and the classifications appears too often, so that the synthesizer does not aim to reproduce the input in all cases. Since the accuracy against the ground-truth is not highly exceeding that of the classifier itself, we can presume that it does not learn to correct errors. More likely is that the classifier has many ambiguous cases which are barely flipping the label during synthesis.

To counter this, in this next experiment, we aimed to retain the classifier's labelling. Instead of using the ground-truth, the target is set as the predicted classifications. In a certain way this represents a perfectly accurate classifier. This is the second use case as explained in chapter 6.

Table 7.2: Results for the Basic Synthesizer Without Additions Trained on Predicted Classifications

Dataset	Classifier Acc.	Synth. against ground-truth	Synth. against classifications
MNIST	0.9929	0.9925	0.999
CIFAR-10	0.8276	0.828	0.9905
CIFAR-100	0.5613	0.5606	0.9565

In comparison to the first experiment, table 7.2 shows the results with the target set as the predicted classifications. The classifier accuracy stays identical of course and the accuracy against the ground truth is approximately the same as before as well. It is also

still bounded by the classifier's accuracy. However, with this method the synthesizer only learns to retain the input classification. It achieves that goal with extremely high accuracy with a simple, two layer architecture within less than five training epochs.

The results can be explained because any initial errors that the classifiers makes in their predictions do not punish the synthesizers during training. The synthesizer can be fully confident in the input and does not attempt to correct any mistakes, and instead solely reproduces the utility.

Where applicable and not mentioned otherwise, all following synthesizers have been trained and evaluated using the second method.

Let's now look at the generated images. As the goal of the synthesizer is to retain the likelihood vector, we omitted any labels. We are only interested how close the reconstruction is and what the synthetic images look like.

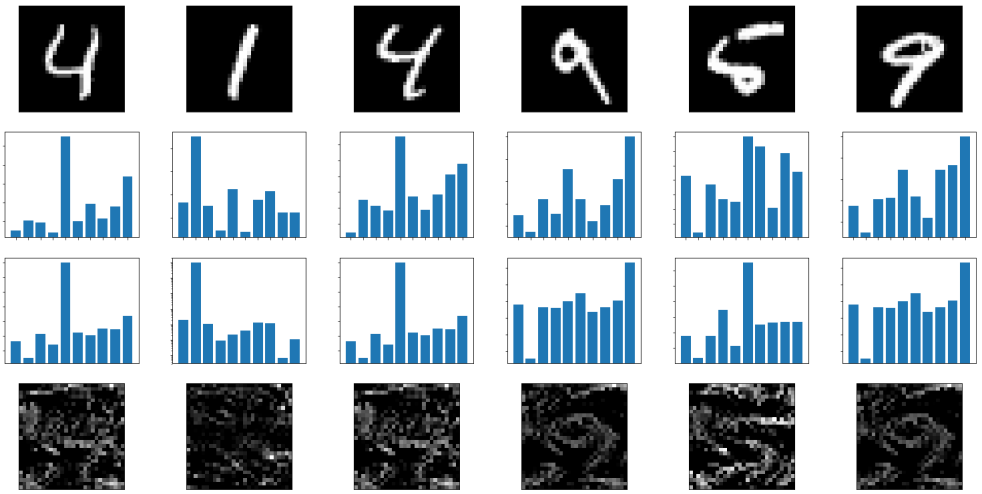

Figure 7.3: Example results of the synthesizer on MNIST inputs. From top to bottom: Original image, the likelihood vector of the original image (in log scale), the likelihood vector of the synthetic image (in log scale), the synthetic image

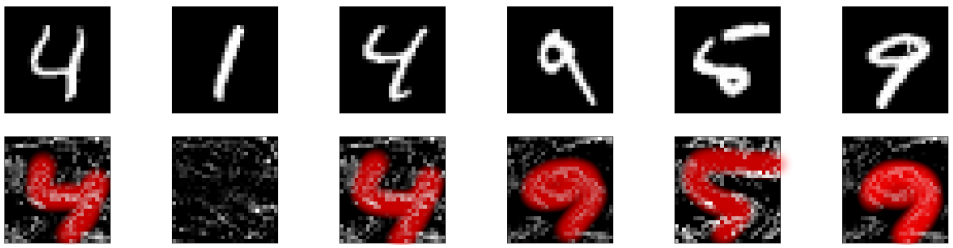

Figure 7.4: Highlighted Shapes in the Synthetic Images

Figure 7.3 shows the resulting synthetic images on the MNIST dataset. The seemingly high variance of likelihood vectors is heavily amplified by the logarithmic scale. It demonstrates an interesting aspect of the synthesizer: in case of handwritten digits in grayscale, it seems that very little information about the shape of the digit is redundant. Many of the numbers still have a faint outline in the synthetic image as highlighted in figure 7.4. This effect is likely enhanced through the use of a convolutional network instead of a fully dense network for the classifier. The convolutional layers are generally more likely to learn edge-detection like features. That means that the classifier uses the edges and outlines to determine the class, and as such those features have to be retained.

Further it becomes apparent that each class effectively has one representative. Even when the input vectors differ, as for example with the two images of a '4', the resulting image and vector are nearly the same visually.

Next we have the resulting images of the CIFAR-10 classifier.

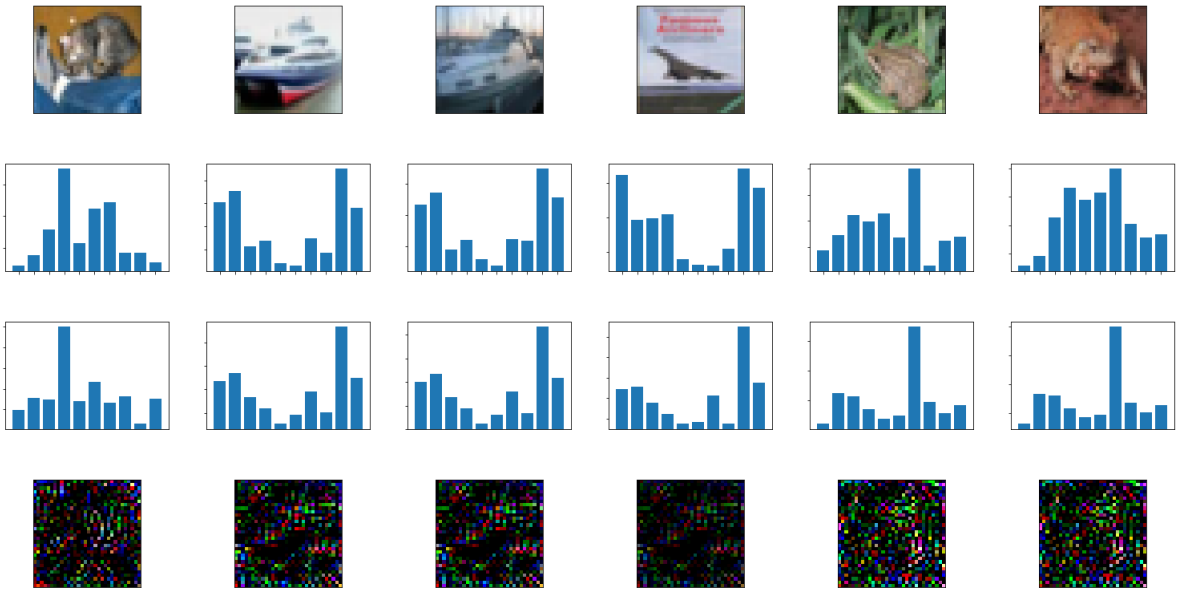

Figure 7.5: Example results of the synthesizer from CIFAR-10 inputs.

More complex images, however, show another kind of pattern. The synthetic CIFAR-10 dataset as seen in figure 7.5 appears to be more random in its nature. It is important to note, however, that the reconstructed image is technically a reshaped vector with 32x32x3 values. There are different ways to reshape such a vector of course, which would lead to different shapes and colours of outputs. Additionally, the images are not normalized in any way. Such normalization would have to occur during synthesis. Once again though, the images two and three, and four, all classified as 'boat', show a visually similar output.

The fourth synthetic image appears to be more faint than the other two, but with similar shapes.

7.4 Multi-Function Utility Synthesizer

This synthesizer was trained on the CIFAR-100 dataset while using both the CIFAR-10 and CIFAR-100 classifier.

Table 7.3: Results for the Multi-Function Utility Synthesizer

Dataset	Classifier Acc.	Synth. against ground-truth	Synth. against classifications
CIFAR-10	-	-	0.9698
CIFAR-100	0.5613	0.5601	0.9226

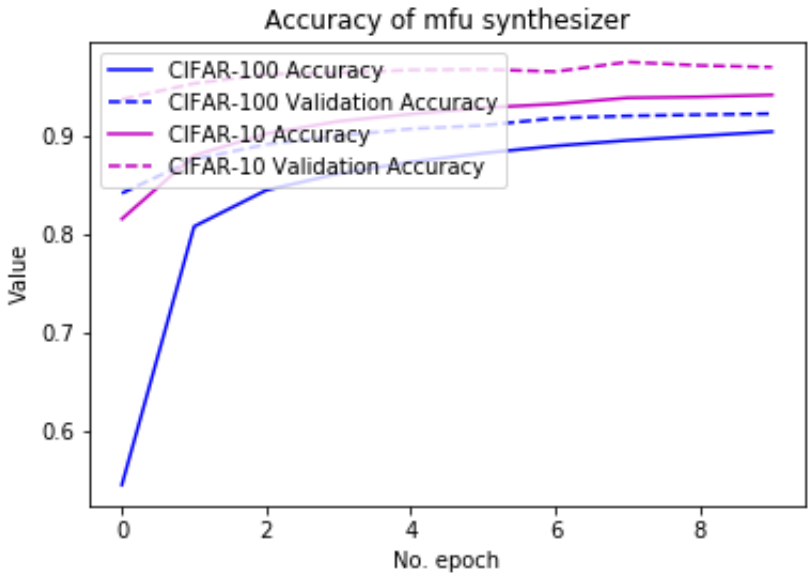

Figure 7.6: Training curve of the Multi Function Utility Synthesizer.

In table 7.3, since the target for the CIFAR-10 classifier were its own predictions, there is no value for its general accuracy. The comparison against the ground-truth is inapplicable as well, as it is identical in concept to the accuracy against the classifications since the ground-truth in this case is the classifications.

While we can see that the accuracy is still rather high, there is a clear drop. The more complex CIFAR-100 classifier is considerably less accurate with roughly 92% than the CIFAR-10 classifier with nearly 97%. Both values fall beneath the accuracy of their re-

spective basic synthesizers. The training took a bit longer, reaching its peak around three epochs in, as seen in figure 7.6.

Considering the simplicity of the network, and the complexity of the classifiers, these are still highly successful results. The CIFAR-100 classifier is complex by design, and the CIFAR-10 classifier has additional complexity as it is originally trained on a different dataset.

The implementation details can be found in the appendix 10.2 10.2.

7.5 Likelihood Vector Retaining Synthesizer

Next up, we look into the likelihood vector retaining synthesizers. As mentioned before, we use our classification problems in this case to analyse the synthesizer's potency against regression or multi-label problems. Instead of reconstructing a single label, we aim to keep the full likelihood vector for all classes. Obviously, keeping the full vectors is worse for privacy than just rebuilding the correct label, one-hot-encoding the results, as we've done in the previous experiments. However, we change the goal here and focus more on the viability of the synthesizer than the specific use-case we have with the image datasets.

For this experiments, we also included a deeper CIFAR-100 synthesizer with more hidden layers 10.2.

We have another metric aside from the accuracy, the mean absolute error 3.10. The mean absolute error indicates the total difference between the original the synthetic utility vector. These values are in particular useful to compare how the synthesizer manages on different datasets.

Table 7.4: Results for the Likelihood Retaining Synthesizer Without Additions Trained on Predicted Classifications

Dataset	Classifier Acc.	Synth. vs. gt	Synth. vs. classif.	Mean Abs. Error
MNIST	0.9929	0.9921	0.9981	0.001307
CIFAR-10	0.8276	0.8276	0.9732	0.02076
CIFAR-100	0.5613	0.5573	0.9186	0.00508
Deeper CIFAR-100	0.5613	0.4982	0.8256	0.00849

As we can see from table 7.4, the accuracy gets slightly worse compared to the basic synthesizer. The CIFAR-10 synthesizer reaches an accuracy of 97.32% against the classifications, ca. 2% worse than with the basic synthesizer. This shows a higher complexity in this task. We are still using a very simple model architecture, however.

The mean absolute error allows us to compare the MNIST and CIFAR-10, and the two

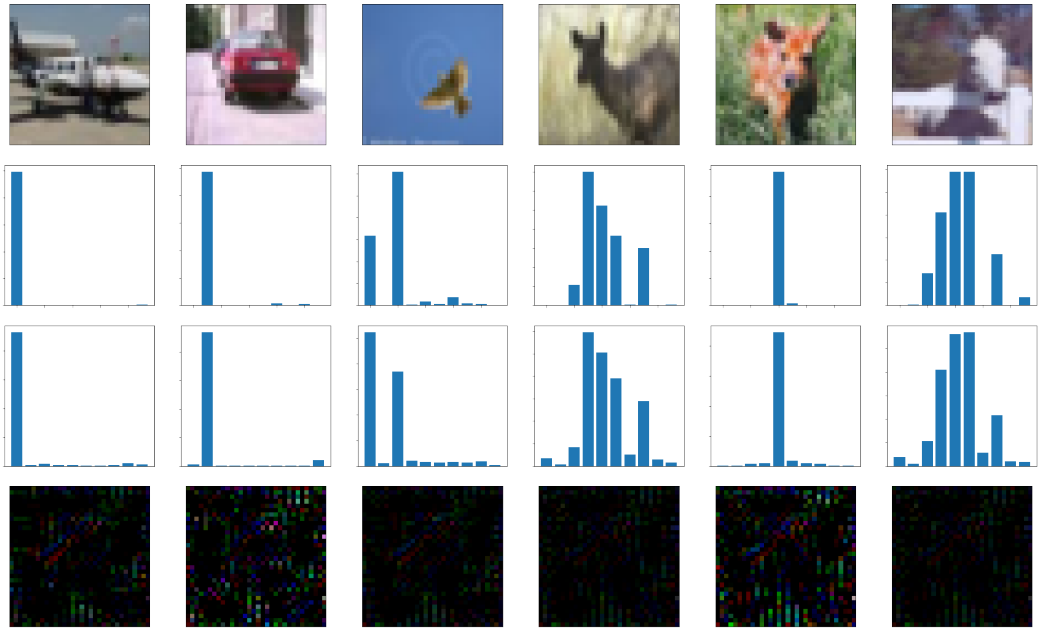

Figure 7.7: Example results of the likelihood retaining synthesizer on CIFAR-10. The vectors are in linear scale.

CIFAR-100 models to each other respectively. We can see that the MNIST dataset yields a low overall error. Since the classes are not ambiguous for the most part, this result was expected. In comparison, the CIFAR-10 dataset leads to an error 15 times higher. It is still rather small overall.

With CIFAR-100, the overall error is smaller, as the differences split over 100 classes instead of only ten. This leaves a bit more room for error in each bar without impacting the error as much.

The deeper CIFAR-100 model shows that the depth does not necessarily correlate with the accuracy, and that the simple network we use is not bad in comparison. A detailed tuning of hidden parameters was not within the scope of this thesis though, this was merely intended as a small aside.

In figure 7.7 we see different images from a synthesizer aimed at retaining the likelihood vector. The y-axis of the vectors is not log-scaled to give a clearer representation of the actual absolute differences.

Here we can see a potential reason for the reduction in the accuracy. Since only the eventual label counts in this metric, small inaccuracies can lead to stark differences in accuracy. As the last image in that series demonstrates, a, in absolute values, small difference between the reconstructed and the original likelihood vector could easily lead to just barely

flipping the label, decreasing the accuracy. In other cases, in which the classifier might be unsure between two or more different labels, this effect could occur as well. However, in this case table 7.4 shows that the decrease in accuracy goes along with a small increase of the mean absolute error, meaning that the synthesizer generally performs worse.

The overall results are still successful. The results on the very simple MNIST dataset are promising. With more fine tuning of the hyperparameters of the networks, even better results could be achieved for the other datasets as well.

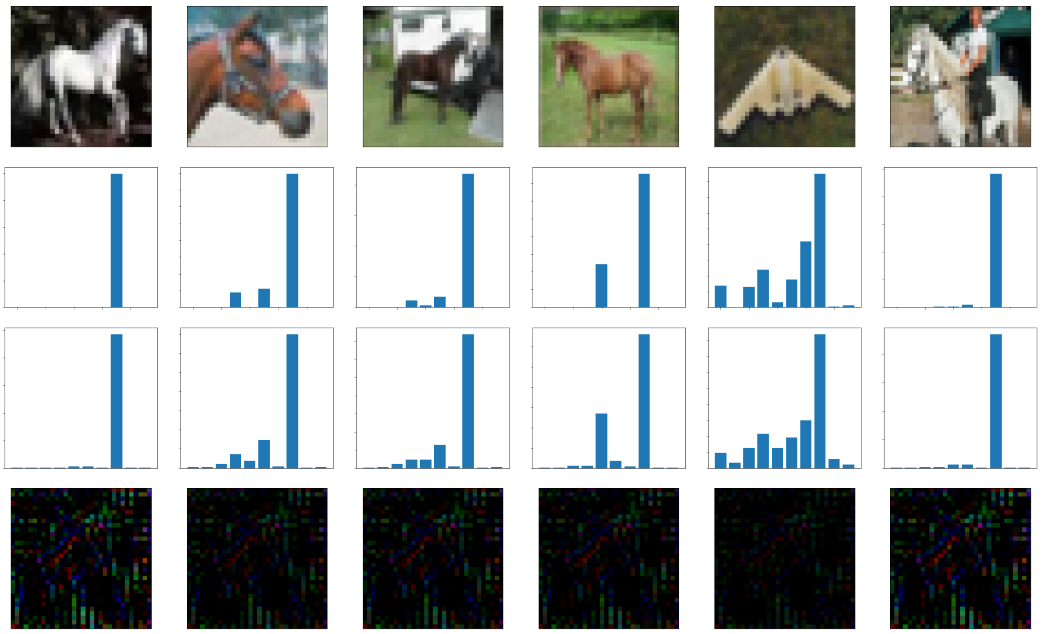

Figure 7.8: Example results of the likelihood retaining synthesizer on CIFAR-10 images. All input images were classified as 'horse'.

A particularly interesting result can be seen in figure 7.8. The difference between different type of horse pictures (head only, from the side, with rider etc.) seem to lie mainly in the intensity of the synthetic image's colors. The faint shape appears to be a common representative for the class 'horse', no matter the type.In general, the effect might also depend on the classifier, however. Seeing that most of these show a very confident vote for the class 'horse', the classifier takes its part in ignoring the other factors, like perspective and rider. But with the second to last image, we can still see that effect. While the input vector greatly differs from the other ones, the general shape is still faintly visible. Even in this use case, the synthesizer creates basic representatives for the classes.

7.6 Noisy Likelihood Vector Retaining Synthesizer

For the noisy synthesizer, we analyse the effect of Gaussian noise on the accuracy of the synthesizer. Adding noise allows us to improve the privacy. Instead of synthesizing the images from the original vector, we add small amounts of noise to that input. This random addition or subtraction on each bar of the vectors leads to the perturbation of the code space. The effect of that noise, how much we can and need to add, two additional factors: The closeness of each cluster of classes to the other clusters, and the closeness of vectors *within* each class.

Table 7.5: Results for the Noisy Likelihood Synthesizer with Guassian Noise with a 0.1 standard deviation.

Dataset	Synth. vs classif.	Synth. vs noisy classif.
MNIST	0.9981	0.9973
CIFAR-10	0.9732	0.9597

Table 7.6: Evaluation of Distances within the Classifications

Dataset	Mean Abs. Distance (MAD)	RMSE	Min. difference	Mean difference
MNIST	0.00276	0.00582	0.00148	0.98720
CIFAR-10	0.04153	0.07728	0.00030	0.76434

Table 7.5 compares the accuracy without noise to the accuracy with noise. We used Gaussian noise with a standard deviation of 0.1. As expected, the accuracy sinks a bit. The effect is weaker on MNIST than CIFAR-10, however.

The explanation for that lies within table 7.6. These results are independent of the synthesizer, and solely present properties of the classifiers. The mean absolute distance and the root mean squared error (RMSE) are the mean distance between images of a single class[2]. The RMSE gives more wight to outliers, and as such is generally higher than the MAE.

The minimal and mean difference describe the difference between the most likely and the second most likely class within a single vector. This gives us an indicator of the confidence of the classifier. A value of 1 would be a confident classifier, having a single class with 100 percent probability. The minimal distance show the most ambiguous case. Even with MNIST, our classifier has at least one case where the difference between two vectors is nearly 0, they are almost equally probable.

[2]Due to computational limits, we decided to randomly choose 1000 pairs of vectors per class instead of doing a pair-wise computation for all elements of each class.

Overall we can see, that the MNIST classifier is more confident than the CIFAR-10 classifier. We can also see that the cluster of each class is more spaced out in the CIFAR-10 code space due to higher MAD and RMSE values.

In relation to the noise, this gives us interesting results. When we add, on average, noise of the RMSE value or more, we might end up at a likelihood vector resembling more that of another image of the same class. At the same time, adding noise beneath the mean difference will generally not change the class, keeping up the accuracy. Of course, these are mean values, meaning that there might still be outliers.

These metrics can be used to determine the right amount of overall noise to add, which might be especially interesting in real life use cases. The exact details depend on the type of noise used and the goals that one might want to achieve.

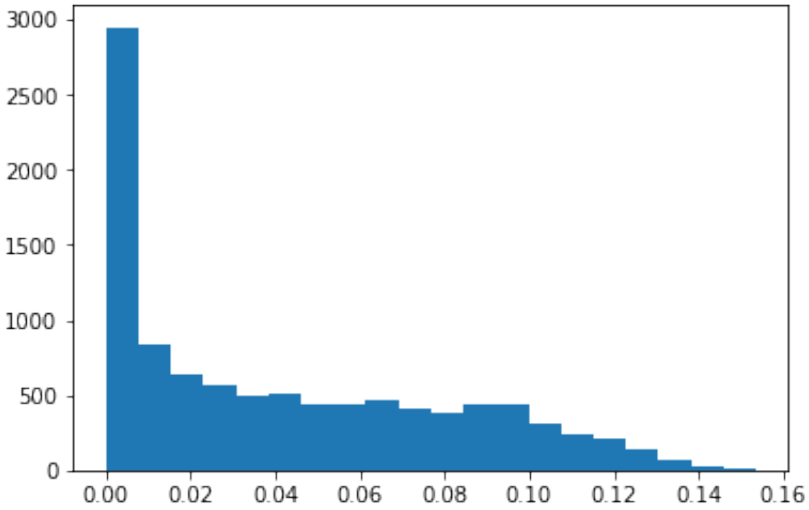

Figure 7.9: Histogram of the Mean Absolute Distances between the Pairs in the CIFAR-10 classifier over 20 buckets.

Figure 7.9 illustrates the mean absolute distances in the CIFAR-10 classifications in form of an histogram. We polled 1000 pairs per class, 10.000 pairs overall. Histograms like that can further be used to determine the effect of noise. We can see that, in most cases, the mean absolute distance is between 0 and 0.1, which means that low amounts of noise can already be enough to blur the code space to an attacker, meaning that they could not clearly determine which input image the vector belongs to.

The images produced through noisy synthesis 7.10 show no unexpected results. The noise mainly affects the intensity of the colours. We could predict that result, as we could see in figure 7.8 that, as long as the predicted label doesn't change, the difference mainly lies in the colour intensity.

Figure 7.10: The effect of noise with a standard deviation of 0.1 on the outputs of a CIFAR-10 synthesizer.
Top to bottom: Original Image and utility vector, regular synthetic image and utility vector, noisy synthetic image and utility vector

8 Limitations and Future Research

In this chapter we look into the limits, both of the significance of the results and of the approach as a whole, and discuss potential research fields for future works. We discuss the limits of experimental proofs of concepts and the loss of future potential of the dataset after the synthesis. We evaluate the computational requirements and how they could be mitigated. Lastly, we analyse the relaxed privacy requirement compared to full differential privacy.

8.1 A Proof of Concept

A major part of this thesis lies in the experimental results. While those seem to be promising on the viability of the approach in and of itself, they are still just a proof of concept. It is not possible to safely deduce from these limited tests how the synthesis will work under different circumstances, with different kinds of data and other utility functions.

Future works could study the synthesizer for example on true regression problems or with a utility function from a production system. Further we have not deeply investigated the effect of deeper or differently built models for the synthesiser. The tuning of those hyper parameters is one point that could improve the accuracy of a given system.

8.2 Loss of Future Potential

Beyond the results, there are also conceptual effects our method of achieving contract-limited privacy has. While it assures that an attacker cannot gain any information beyond potentially the label or utility vector, the same goes for the legitimate user. Once the synthesis is run, the original data should be disposed of. This leads to the requirement that all the types of information that are wanted have to be known beforehand. This potentially limits the real world use cases, as it might be that a user does not fully know how the information will be processed. At the same time, the original data owner could be sure that the information will only be used in the way that has been agreed upon. This additional assurance might put the user more the user more at ease when handing over personal data for processing.

A way to act against that loss of potential would be to store the original datasets securely and offline, using the synthesis in cases where further processing has to be done off premise. For example as user could create a synthetic version of the data they wanted to transmit locally, before sending the synthetic data off. This would work without having to adapt the system, since all pre-processing happens client-side. This approach would likely have to assure, however, that the synthesis is not computationally expensive.

8.3 Computational Requirements

The synthesis has considerable computational requirements. As of now, the system works by running the full utility function before it can synthesize the data. While that might not be a problem with small image classification, the effects might be considerably larger on real world legacy systems, up to the point where synthesizing data off premise might not be feasible.

One potential way to counter that would be to train another synthesizer, but this time as a student in a teacher-student-network. The idea is that they student network learns to mimic the results of the teacher at a fraction of the size. The student does not have to know how the classification function works, it just has to create the right synthetic images based on an input. Instead of working against each other like an adversarial, the student gets trained on the synthetic dataset of the teacher. This approach could reduce the dimensionality of the original synthesizer and would be an interesting field for future research.

Other Knowledge distillation techniques could be used to create a new network that learns from a highly complex network that preserves, for example 100 utility functions. This new network could be smaller and more manageable while keeping the accuracy. Research in the area of knowledge distillation[HVD15] could be applied to the synthesis described in our work.

8.4 Relaxed Privacy Requirement

While contract-limited privacy is achieved, we do not gain differential privacy with our approach. In comparison to k-anonymity for example, contract-limited privacy is weaker. Due to the lack of groups of a minimal size greater than one, individuals could still be singled out if they are the only representative of their class, or an attacker has sufficient background knowledge to narrow it down to single people. This is not a problem in regards to the GDPR, but it has to be kept in mind.

9 Conclusions

Our goal was to to build a data synthesizer for use with a legacy system that achieves contract-limited privacy. Beyond that, we tested the use of such a synthesizer with different additional challenges imposed on it. We analysed a possible way to improve the accuracy of the synthesizer and have proven why that was not feasible without potentially introducing considerable privacy leakage.

Through our experiments, we were able to show the viability of a data synthesizer as a means to achieve contract-limited privacy. Our proof of concept was able to synthesize images that would uphold the utility of the classifiers with over 99% accuracy even on the CIFAR-10 dataset. Beyond the basic synthesizer, we could further demonstrate the versatility of the approach by applying different additions. Keeping the utility of multiple classification functions was shown to be successful in principle, with still high accuracies of more than 97% and 92% respectively to each synthesizer. Our simulated regression problem has proven its potential as well. Upholding the likelihood vectors with high accuracy opens up the possibility for promising further research on actual regression datasets. We further demonstrated the option of adding noise during the synthesis. This increases the viability in regards to privacy.

It is important to note that these results are not conclusive in regards to possibly achievable accuracies. We specifically build a simple proof of concept, mostly disregarding the options of tuning hyper parameters, like the amount, types and settings of hidden layers.

Especially notable was the simplicity of the final synthesizers. With merely two hidden dense layers, the training was accordingly quick. It mostly took less than 5 epochs to reach the final accuracy. This could mean that larger, more complex datasets could be handled simply by increasing the depth of the synthesizer. As it is usual for most things related to neural networks though, there is no one-size-fits-all solution. The exact architectures likely have to be adapted to any given network, utility function, goal and other aspects.

10 Appendix

In this chapter are additional graphs and model architectures for reference. This chapter contains technical terms that are not explained in this thesis and mainly presents reference points for the potential reproduction of the results.

10.1 Classifiers

1. Batch Size: 64
2. Activation: elu in hidden layers, softmax at the final layer
3. Optimizer: adam

10.2 Synthesizers

1. Batch Size: 64
2. Activation: relu in hidden layers, softmax at the final layer
3. Optimizer: adam

10.3 Training Curves

```python
def model_build():
    inputs = tf.keras.Input(shape=(img_width, img_height, img_channels), name='inputs')

    x = Conv2D(8, kernel_size=(3,3), activation='elu')(inputs)
    x = BatchNormalization()(x)
    x = Conv2D(16, kernel_size=(3,3), activation='elu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Dropout(0.2)(x)

    x = Conv2D(16, kernel_size=(3,3), activation='elu', padding='same')(x)
    x = BatchNormalization()(x)
    x = Conv2D(16, kernel_size=(3,3), activation='elu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Dropout(0.2)(x)

    x = Conv2D(32, kernel_size=(3,3), activation='elu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Dropout(0.4)(x)
    x = Flatten(name='flatten')(x)

    output = Dense(num_classes, activation='softmax', name='denseOut')(x)

    model = tf.keras.Model(inputs=inputs,
                           outputs=output,
                           name=('classifier_'+desc+'_'+train_dataset))

    model.compile(optimizer='adam',
                  loss=losses.CategoricalCrossentropy(),
                  metrics=['categorical_accuracy'])
```

Figure 10.1: The General Architecture of the Classifiers, adapted as needed to fit the input format

```
def model_build(classifier1, classifier2):
    inputs = tf.keras.Input(shape=(img_width, img_height, img_channels, ),
                            name='inputs')

    a = classifier1(inputs, training=False)
    a = Dense(32)(a)
    b = classifier2(inputs, training=False)
    b = Dense(32)(b)

    c = layers.concatenate([a, b])

    x = Dense(256, activation='relu')(c)
    x = Dense(64, activation='relu')(x)
    x = layers.Dropout(0.4)(x)

    imageOutput = Dense((img_width*img_height*img_channels),
                        activation='relu', name='denseOut')(x)

    imageReshape = Reshape((img_width, img_height, img_channels),
                           input_shape=((img_width*img_height*img_channels),),
                           name='reshape2')(imageOutput)

    classification1 = classifier1(imageReshape, training=False)
    classification2 = classifier2(imageReshape, training=False)

    model = tf.keras.Model(inputs=inputs, outputs=[classification1, classification2],
                           name=('synthesizer_classif_'+desc+'_'+train_dataset))
    model.compile(optimizer='adam', loss=losses.CategoricalCrossentropy(),
                  metrics=['categorical_accuracy'])

    return model
```

Figure 10.2: CIFAR-100 Multi Function Utility Synthesizer

```
def model_build(classifier):
    inputs = tf.keras.Input(shape=(img_width, img_height, img_channels, ),
                            name='inputs')

    x = classifier(inputs, training=False)

    x = Reshape((10,5,2))(x)
    x = Conv2D(8, kernel_size=(3,3), activation='elu')(x)
    x = BatchNormalization()(x)
    x = Conv2D(16, kernel_size=(3,3), activation='elu', padding='same')(x)
    x = BatchNormalization()(x)
    x = MaxPooling2D(pool_size=(2,2))(x)
    x = Dropout(0.2)(x)
    x = Flatten()(x)
    x = Dense(256, activation='relu')(x)
    x = Dense(32, activation='relu')(x)
    x = Dropout(0.4)(x)
    imageOutput = Dense((img_width*img_height*img_channels),
                        activation='relu', name='denseOut')(x)

    imageReshape = Reshape((img_width, img_height, img_channels),
                           input_shape=((img_width*img_height*img_channels),),
                           name='reshape2')(imageOutput)

    classification = classifier(imageReshape, training=False)

    model = tf.keras.Model(inputs=inputs, outputs=classification,
                           name=('synthesizer_'+desc+'_'+train_dataset))
    model.compile(optimizer='adam', loss=losses.mean_squared_error,
                  metrics=['accuracy', tf.keras.metrics.RootMeanSquaredError()])

    return model
```

Figure 10.3: CIFAR-100 Deeper Synthesizer

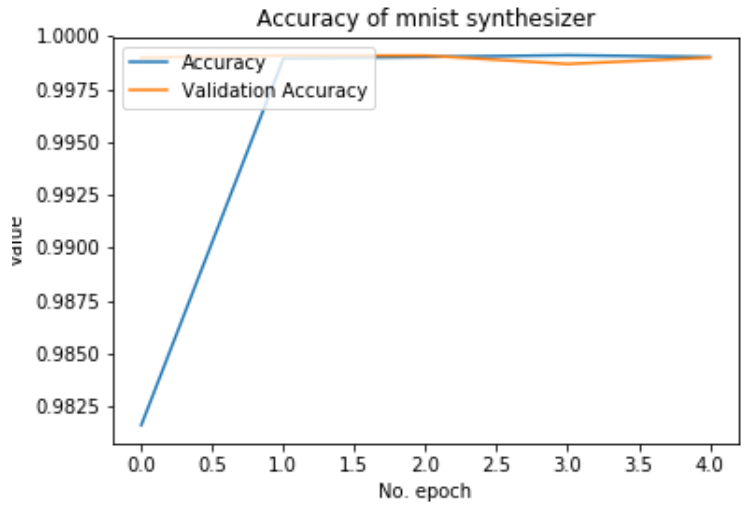

Figure 10.4: MNIST Synthesizer Training Accuracy

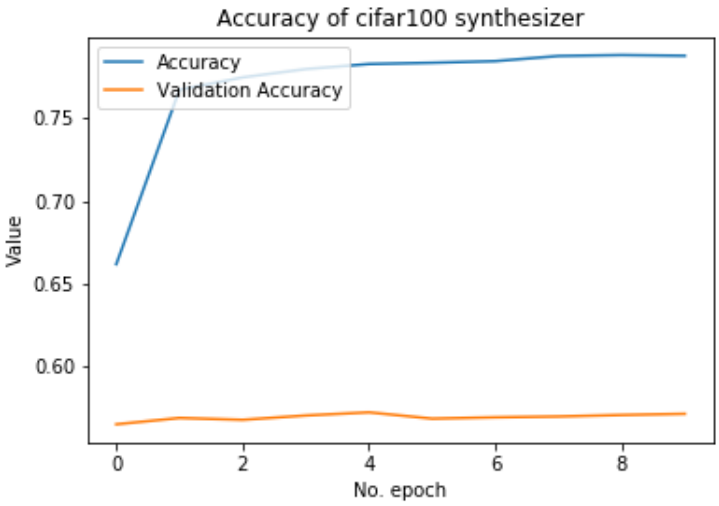

Figure 10.5: CIFAR-100 Synthesizer Training Accuracy

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[aws] Amazon sagemaker ml instance types.

[BSG17] Vincent Bindschaedler, Reza Shokri, and Carl A. Gunter. Plausible deniability for privacy-preserving data synthesis, 2017.

[DR+14] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[gdp04] Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation). 2016-05-04.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *stat*, 1050:9, 2015.

[KH+09] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[LCB10] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. 2010.

[MZ17] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 19–38. IEEE, 2017.

[NS08] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large

sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.

[Ste19] Vincent Stettler. Privacy preserving data synthezisers and utility dense manifolds. Master's thesis, ETH Zurich, 2019.

[Swe02] Latanya Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[WAB+18] Alexandra Wood, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David R O'Brien, Thomas Steinke, and Salil Vadhan. Differential privacy: A primer for a non-technical audience. *Vand. J. Ent. & Tech. L.*, 21:209, 2018.