UNIVERSITÄT ZU LÜBECK
INSTITUT FÜR IT-SICHERHEIT

Analyzing the impact of fine-tuning foundation models on the performance of DP-SGD

Analyse des Einflusses angepasster, vortrainierter Modelle auf die Genauigkeit von DP-SGD

Bachelorarbeit

im Rahmen des Studiengangs
IT Sicherheit
der Universität zu Lübeck

vorgelegt von
Eric Landthaler

ausgegeben und betreut von
Prof. Dr. rer. nat. Esfandiar Mohammadi
M. Sc. Johannes Liebenow

Lübeck, den 09. Dezember 2022

Abstract

Artificial Intelligence (AI) is used in many areas of our lives. They are designed to process data and find solutions to given problems. This requires data and algorithms to train a reliable system. Due to the increasingly widespread use of such artificial models, there is a large number of public data sets that can be used for training. However, there is also data we do not necessarily want everyone to know. So we have to protect them.
To realise that, the principle of Differential Privacy (DP) can be used to ensure security guarantees for certain algorithms. To accommodate these requirements in the field of AI, training-mechanisms can be used that meet Differential Privacy. Differential Private Stochastic Gradient Decent (DP-SGD) is such an algorithm. In this thesis, we want to investigate to what extent fine-tuning of pre-trained models affects the performance of DP-SGD. The motivation behind it is the combination of knowledge. The question arises whether training with different datasets improve the whole system.
In order to be able to investigate this question better, we present a two-step approach. First, we pre-train a model on different data sets. We then use this model to train a linear layer in a differential private manner, measuring the accuracy on an unknown data set. In order to achieve efficient pre-training, we use the contrastive learning approach SimCLR. The great advantage of such approaches is that they do not require labels. This enables us to apply it to numerous data sets.
We will show that differently trained models give similar results for the same task. This is independent of how far or with which data set they were trained. Furthermore, we will discuss and evaluate possible reasons for this.

Künstliche Intelligenzen (KI) kommen in vielen Bereichen unseres Lebens zum Einsatz. Diese sind so konzipiert Daten zu verarbeiten und Lösungen für gegebene Probleme zu ermitteln. Dazu sind Daten und Algorithmen notwendig, um ein zuverlässiges System trainieren zu können. Durch die zunehmende Verbreichtung solcher künstlicher Modelle gibt es eine Vielzahl öffentlicher Datensätze, die für das Training verwendet werden können. Es gibt aber auch Daten, die nicht zwangsläufig allen bekannt sein sollen. Diese müssen also geschützt werden.
Um dies zu realisieren kann das Prinzip der Differential Privacy (DP) verwendet werden, um Sicherheitsgarantien für Algorithmen zu gewährleisten. Um diesen Anforderungen im Bereich der KI gerecht zu werden, können Trainingsmechanismen eingesetzt werden, welche Differential Privacy erfüllen. Differential Private Stochastic Gradient Decent (DP-SGD) ist ein solcher Algorithmus. In dieser Arbeit wollen wir untersuchen, inwiefern die Spezifikation vortrainierter Modelle die Leistung von DP-SGD beeinflusst. Die Motivation dafür steckt in der Kombination von Wissen. Dabei stellt sich die Frage, ob das Training mit unterschiedlichen Datensätzen das Gesamtsystem verbessert.
Um dieser Frage besser nachgehen zu können, stellen wir einen zweistufigen Ansatz vor. Zuerst trainieren wir ein Modell auf verschiedenen Datensätzen vor. Anschließend verwenden wir dieses Modell, um eine lineare Schicht differential private zu trainieren. Dabei messen wir die Genauigkeit auf einem unbekannten Datensatz. Um ein effizientes Vortrainieren zu erreichen, verwenden wir den contrastive learning Ansatz SimCLR. Der große Vorteil solcher Ansätze ist, dass sie keine Label benötigen. Dadurch können wir diese auf zahlreiche Datensätze anwenden.
Wir werden zeigen, dass unterschiedlich trainierte Modelle für die gleiche Aufgabe ähnliche Ergebnisse liefern. Dies ist unabhängig davon, wie weit oder mit welchem Datensatz diese trainiert wurden. Darüber hinaus werden wir mögliche Gründe dafür diskutieren und auswerten.

Erklärung

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

Lübeck, 09. Dezember 2022

Acknowledgements

I would like to thank Esfandiar Mohammadi for his support and expertise while processing this work. Also, many thanks to Johannes Liebenow, who supported me with technical questions and other matters related to the implementation. Thanks also to Sophia Marcusson, Jorge Andresen and Erik Schmidt for taking the time to proofread this work.

Contents

1 Introduction **1**

2 Preliminaries **3**
2.1 Machine Learning . . . 3
2.2 Contrastive Learning . . . 3
2.3 ResNet Architecture . . . 4
2.4 Differential Privacy . . . 5
2.5 Differential Private Stochastic Gradient Decent . . . 7
2.6 Related Work . . . 8

3 Background **11**
3.1 SimCLR . . . 11
3.2 Tensorflow DP-SGD . . . 12
3.3 T-distributed Stochastic Neighbour Embedding . . . 12

4 Problem Statement and Challenges **13**
4.1 Problem Statement . . . 13
4.2 Approach Overview . . . 14
4.3 Challenges . . . 14

5 Approach **17**
5.1 Method . . . 17
5.1.1 Pretraining . . . 17
5.1.2 Fine-tuning . . . 19
5.2 Implementation Details . . . 20
5.2.1 DP-SGD . . . 20
5.2.2 Custom Datasets . . . 22

6 Experiments **25**
6.1 Technical Overview . . . 25
6.2 Experimental Setup . . . 26
6.2.1 CelebFaces Attributes Dataset . . . 26
6.2.2 Cross-Age Celebrity Dataset . . . 26

6.2.3 IMDB-WIKI Dataset . . . 26
6.2.4 Labeled Faces in the Wild Dataset . . . 26
6.3 Experiments . . . 27
6.4 Experimental Results . . . 28
6.4.1 Model Accuracy . . . 28
6.4.2 Differential Privacy Guarantees . . . 31

7 Evaluation **33**
7.1 Interpretation . . . 33
7.1.1 TSNE . . . 33
7.1.2 Training Curve . . . 35
7.1.3 Evaluation with IMDB dataset . . . 37
7.1.4 Implementation without DP-SGD . . . 37
7.1.5 Supervised Learning on Resnet50 . . . 38
7.2 Discussion of the challenges . . . 39

8 Conclusion **43**
8.1 Summary . . . 43
8.2 Future Work . . . 44
8.2.1 Improvement Pre-Training . . . 44
8.2.2 Improvements DP-Optimizer . . . 44
8.2.3 Improvement Base Model . . . 44

References **45**

1 Introduction

The demand and scope of Artificial Intelligence (AI) is constantly increasing. In many areas, such as autonomous driving, medicine or image processing in general, working without the support of such mechanisms is no longer conceivable today. As a large and important part of AI, machine learning serves as a way to train reliable and efficient models that can independently develop solutions to problems.
For this reason, work has been going on for many years to develop learning methods that make it possible to create such models. Different types of training are used for this, for example self-supervised techniques. The advantage here is that any labelled data is not needed, since self-supervised mechanisms generate artificial labels by creating positive and negative pairs in each batch. Therefore, such techniques are useful in many areas to train accurate models. One method that is widely used is the contrastive learning approach SimCLR [CKNH20] which can be used to train models for image processing. The idea is that you create positive pairs by image transformation. Since the altered images are both based on the same picture, it can be ensured that they are the same and therefore form a positive pair. This approach is the recent state-of-the-art contrastive learning approach.
Therefore, it makes sense to use SimCLR to train a reliable and accurate model that can be used for image processing. Fulfilling this task well is a very essential part of working with artificial intelligence and has a very wide range of applications. However, it may also be the case that we want to classify data that contains sensitive content that is worthy of protection. In this case, we need to use mechanisms that allow us to train models in such a way that the data is protected and there is no way to extract it. The basis for this is the principle of Differential Privacy (DP) [DR14], which enables us to determine security guarantees for algorithms. DP approaches can thus be used to protect the data.
In order to be able to apply this idea in the machine learning area, a method is necessary that allows us to protect the data within the model and to avoid publication of it, even after the training has ended. For these, optimizers that introduce mechanisms to ensure Differential Private training can be used. Differential Private Stochastic Gradient Decent (DP-SGD) [ACG+16] is such an optimizer and contains gradient clipping and noising to restrict the impact of single gradients to the model. As a result, the influence of individual data points is less strong and the privacy of the data can be better protected.

In this work, we will present and implement a pipeline for image classification. For this, we will introduce a two-step approach. First, we will use SimCLR to train a system that can ensure a high accuracy on a provided task for a sensitive data set. We want to investigate, how further SimCLR training affects the accuracy of the model when we apply it to an unknown dataset. Since the used data can include sensitive data, we need to protect them. For this, we will introduce a Differential Private fine-tuning which uses the pre-trained model and train it for the given task. By doing this, we want to ensure that we can create an accurate and reliable model by training on public data, which can be fine-tuned to receive good results for any unknown task.
In order to give a rough overview of the concepts and methods on which we build, we want to introduce contrastive learning and SimCLR in chapter 2, present the model architecture used and explain the concept of DP and the functionality of DP-SGD. In part 3 there will be a short presentation of the code respectively the libraries we are going to build on. After that, there will be a discussion about the problem that is considered in this work and some challenges that could appear during the work. Later on, we want to introduce the pipeline and discuss the code changes, so that the used approaches meet our requirements. This is followed by the presentation of the datasets we want to use and some experiments, followed by their evaluation in chapter 6 and 7. In the end, we want to summarize our findings and show up some open problems, that can be solved in future work.

2 Preliminaries

In this chapter, we will introduce some basic information about machine learning techniques and the idea of differential privacy, which is needed to better understand the following experiments and this work in general.

2.1 Machine Learning

The need for reliable machine learning models and the development of these increases year after year. In areas like medicine, autonomous driving and other fields of our life, we can not imagine to work without such models. The idea of machine learning is to train a model with the target of finding solutions to problems independently. This is done by recognizing patterns in given data. For this, examples and algorithms are necessary, so that the system is able to extract properties of the data, in order to generalize them and to be able to use them later in problem-solving or analysing unknown data.
As a part of the large subject area of Artificial Intelligence, there are many ways to train such models, for example supervised, unsupervised and semi-supervised training techniques. On the one hand, in the supervised case, pairs of in- and output data are provided, and the model has to learn relationships in order to later be able to establish associations with unknown data. On the other hand, in the unsupervised scenario, the model only receives input data, which contain categories and contexts. In this case, the network creates classifiers independently, according to which it divides the input patterns. Semi-supervised training describes a mixture between the supervised and unsupervised technique. In this case, the output is only known for part of the input data.

2.2 Contrastive Learning

Contrastive Learning describes a self-supervised machine learning technique which is used to learn general features of given data. For this, the model has to figure out which data points are similar and which are different, to extract high level features. Therefore, self-supervised mechanisms generate artificial labels by creating positive and negative pairs in each batch.
This type of training is very powerful, because no labelled data is required. Such learning techniques are useful in many areas of application, because you do not need specialists

and experts to analyse and label the training data given to the model. This saves time, money and other resources.
In this work, we will focus on the recent state-of-the-art contrastive learning approach, SimCLR [CKNH20]. This method is used for visual representation tasks, like image classification. For this, each picture gets transformed by two augmentation combinations, like recolouring, resizing and more. During the transformation, it can be ensured that these pictures are similar and can be used to train the model, since both have the same image as a basis. Such pairs are also called positive pairs. As a base for such models, a ResNet architecture can be used. This will be discussed later in section 2.3. In the ResNet itself, a vector represents an image. So at the end the model should create similar representations for related images. Those pictures should be represented by a close vector. Thus we want to maximize the similarity of those two representations of the same image by minimizing the contrastive loss function. Trying to minimize this function means that each positive pair should stay nearby and the negative pairs should be far away. This part of the training is called pretraining.
After pretraining, the model learned high level features of the images. If we want to perform a specific classification task, we now can fine-tune the model with supervised data and improve the accuracy of the model for this specific task.

2.3 ResNet Architecture

To provide a base for our model, we need a specific architecture on which we can perform our training steps efficiently for high dimensional data, in this case images. The problem with deep neural networks is that more layers do not automatically lead to better results. On the contrary, the Learning decreases with increasing depth and it is possible, that this training effect can be reversed and the model is getting worse. The reason for this is the disappearance of the gradients, which are normally used to calculate the adjusted parameters in the training step. If these gradients are lost, there can be a negative impact on the model itself. One effect that could appear, is that the data is memorized and the model unlearns how to generalize.
To avoid this difficulty, Residual Networks [HZRS15] can be used. The idea is, that we do not want the layers to learn the final function to the issue our model should solve. Instead of this, the authors use a residual network at the end of the layers to learn the residual function. So the idea is, that each layer learns new and different features and lead them to the residual network. Hence, the approach is that the accuracy of the whole model can be increased, when the model is able to learn a difference between in- and output. If this difference can be minimized, it is possible to find an optimal solution. To obviate the

disappearance of the gradients, shortcut connections are provided. These ensure that the gradients do not pass weight layers, if the input- and output dimension of the layers are equal, so the model can not extract new information. In this case, the gradients are not changed, so they can not disappear. This leads to good training results.
In case of using this model architecture, it is possible to vary the number of layers that are used. In this work, we will focus on the ResNet 50 architecture. So we use a neural network with 50 layers. This type of Residual Network achieves good results in image processing.

2.4 Differential Privacy

Data is an important commodity of our time and protecting this is one of the central tasks in various areas. In the Netflix Price Competition from 2006, we have seen how an anonymised dataset was given and how it was possible to reconstruct personal information by adding public data to the original. That showed how simple mechanisms can not ensure the privacy of unique data points. But in some areas, for example in the medical field, it must be guaranteed, that we can provide the anonymity of single points in our dataset. However, there is a trade-off between the anonymity and the use of the data after anonymization.
To meet this requirement, Differential Privacy (DP) [DR14] can be used. The approach of DP is to guarantee, that the privacy of the data can be ensured to a certain extent. In the scenario of machine learning the model should only learns things about the general dataset, but after training it is not possible to reproduce the impact of a single data point to the whole system. So, the goal is to maximize the accuracy of our model by minimizing the probability to identify the impact of a single data point.
In this work, we will focus on the (ϵ,δ)-Differential Privacy. To explain the insurances and the idea itself, we need to define two subsets of a given database. We introduce mechanisms to measure the size of the dataset and make statements, about the data points themselves. For this, we can use the set minus of datasets with the aim of putting two data sets in relation to each other. For that, we want to use datasets as an amount of data points consisting as a tuple of index and the data itself.

Definition 2.1. (Set minus of Datasets). Let D be a dataset with n entries and let each entry x have the form (i, φ), whereby i $\in \mathbb{N}$ and $\varphi \in$ input-space. Let d_1 and d_2 be two subsets of D. The set minus of d_1 and d_2 is denoted as $d_1 \setminus d_2$ and is defined to be:

$$d_1 \setminus d_2 = d_3, \forall x \in d_3 : x \in d_1 \wedge x \notin d_2$$

Notice that the set minus between two datasets d_1 and d_2 with $d_1 \setminus d_2$ only contains the data points, that are part of d_1, but not of d_2. This concept will be very helpful to describe the relation between two datasets, which is needed by the definition of differential privacy.
At last, we need the definition of the cardinality of datasets to determine the size of these.

Definition 2.2. (Cardinality of Datasets). Let D be a dataset with n entries. The cardinality of D is denoted as $|\mathrm{D}|$ and is defined to be:

$$|\mathrm{D}| = \mathrm{n}$$

As you can see, the cardinality returns the number of entries for a given dataset. Furthermore, for two given datasets d_1 and d_2, $|d_1 \setminus d_2|$ specifies the exact number of data points that are part of d_1 but not part of d_2.
Now that we have introduced these two concepts, we can begin to discuss DP and the properties of differential private mechanisms. As we have seen before, the goal of differential privacy is to prevent attackers from being able to distinguish whether a specific data point is contained in a given dataset or not. So it tries to minimize the impact of each data point to the learning process itself. To take a closer look at this goal, let us start with the formal definition of differential privacy:

Definition 2.3. (ϵ,δ)-Differential Privacy). Let $\mathcal{M}$ be a randomized algorithm. We say that a pair of datasets d_1 and d_2 is neighbouring if they differ in at most one element, i.e., $|d_2 \setminus d_1| \leq 1$ and $|d_1 \setminus d_2| \leq 1$. The algorithm $\mathcal{M}$ is (ϵ, δ)-DP if for all neighbouring datasets d_1, d_2, all $\mathcal{R} \subseteq$ Range($\mathcal{M}$), the following holds:

$$\Pr[\mathcal{M}(d_1) \in \mathcal{R}] \leq e^{\epsilon} * \Pr[\mathcal{M}(d_2) \in \mathcal{R}] + \delta$$

This formula describes, that a mechanism that provides (ϵ,δ)-Differential Privacy has to ensure, that an observed value $\mathcal{M}(x)$ is no more or less likely to occur when the dataset is d_1 than when the dataset is d_2. This must hold for each pair of neighbouring datasets d_1 and, d_2 in D. In this case, ϵ describes the maximum distance between a query on dataset d_1 and the same query on dataset d_2. This means, that for a given mechanism $\mathcal{M}$, the output distribution should be nearly the same whether $\mathcal{M}$ is run on d_1 or d_2. Since d_1 and d_2 are not identical the same, this can lead to a change in the distribution. If $\mathcal{M}$ ensures (ϵ,δ)-DP, this change should be very small, since only one data point differs d_1 and d_2. This difference in the distribution can be represented by e^{ϵ}. δ is a parameter which contains the probability of information, that are accidentally leaked. We want to minimize the

leakage of data, because they are sensitive and therefore worth protecting, so we want to choose δ very small. This makes our algorithm $\mathcal{M}$ indistinguishable from an algorithm that randomly selects data points from the given dataset. So at least we want a $\delta < \frac{1}{|D|}$ to get better privacy results than such an algorithm.

2.5 Differential Private Stochastic Gradient Decent

Stochastic Gradient Decent (SGD) is a method to discover the optimal configuration for a machine learning algorithm. For this, it makes small adjustments of the network with the goal to decrease the error and increase the accuracy of the model. To realize the adjustments, Stochastic Gradient Decent tries to find the global minimum of the error by approximating the gradient for batches, which are randomly selected. The random selection can be achieved by shuffling the dataset and pick up the batch out of the result.
As an improvement of the normal SGD implementation, Abadi, Chu, et al. introduce Differential Private Stochastic Gradient Decent, called DP-SGD [ACG+16]. The motivation to introduce DP to this mechanism is, as discussed in 2.4, to avoid the leakage of train data after the training process. The need for such a privacy preserving optimizer is shown in the work of Ian Goodfellow [Goo15], where he shows how to reconstruct images that were used during the training of a model without any privacy mechanisms like DP. In addition to SGD, DP-SGD provide mechanisms to limit the privacy loss per gradient update. The first approach is to clip the gradients to an upper bound for the l_2 norm. This bound will be called C in the following. After that we can add noise, called σ, in a second step, to the gradient update with a given noise multiplier. By editing C and σ, we can have an impact on the (ϵ,δ)-Differential Privacy guarantee our model provides.
To get a general overview about the results of the algorithm, we shortly want to describe the previously shown steps to ensure the privacy of the data, by editing the gradients. At first there is the Norm of clipping. In this step, the gradient vector g is replaced by the normalized vector $\tilde{g}$. For this, the l_2 Norm is used, which is defined as:

Definition 2.4. (l_2 Norm of a vector). Let $\vec{x}$ be a vector of dimension n. The l_2 Norm of $\vec{x}$ is denoted as $||\vec{x}||_2$ and is defined to be:

$$||\vec{x}||_2 = \sqrt{(x_1)^2 + (x_2)^2 + ... + (x_n)^2}$$

It is also known as the Euclidean Norm.

This definition can be used to calculate the normalized vector $\tilde{g}$. This can be realized as follows:

$$\tilde{g} = \frac{\vec{g}}{max\{1, ||\vec{g}||_2/C\}}$$

Notice, that if $||g||_2 < C$ then $\tilde{g}$ equals g, while for $||g||_2 > C$ $\tilde{g}$ is scaled down to be of norm C. This reduces the impact of a given gradient.
For adding noise, the authors used the Gaussian Noise algorithm with a standard deviation given as:

$$\sigma = \frac{\sqrt{2 * log(\frac{1.25}{\delta})}}{\epsilon}$$

By using this standard deviation to calculate the noise, the algorithm provides the (ϵ,δ)-Differential Privacy for each step.
As you can see, at the end of the algorithm, we receive normalized vectors with the added noise. This will help us to reduce the impact of each training step and realize the concept of DP. So this algorithm can be used to carry out the training of machine learning models in a privacy preserving manner.

2.6 Related Work

In this chapter, we will present various works, which focus on different approaches to solve problems using differential privacy respectively contrastive learning. For this, we will shortly present the used method and the followed goals of each work and discuss shortly how our work differs from these approaches.
The authors Li, Yan et al. present in their work about Differential Private Contrastive Learning (DPCL) [LYW+22] a privacy risk analysis of contrastive learning. For this, they implement a mechanism for correlating gradients during the contrastive learning to decrease the sensitivity of each gradient. To evaluate this mechanism, a SimCLR training is used. They show, that their improvement decreases the sensitivity of each gradient and lead to a higher accuracy than the common DP-SGD. While this work mainly focuses on safety analysis and improvement of this, we want to focus on the pure improvement of DP-SGD through further pretraining.
An enhancement of the SimCLR pretraining mechanism is presented by Li, Guo et al. in their publication about Robust age estimation model using group-aware contrastive learning [LGW+21]. In this work, the authors want to extract characteristics from given face images, especially the age of the people. For this, they present a modified version of creating positive and negative pairs. Contrasted with the standard approach where an image is taken and transformed, this approach follows the idea that images of people with the same age should be positive pairs and people with a different age should be negative pairs. So they do not perform two transformations of one picture, they select two pictures from the same class. Later on, this model will be used to solve the regression task to predict the age based on a given dataset. This approach thus pursues the improvement

of the SimCLR pretraining in order to be able to achieve better results in the regression step, while we want to achieve improvements not through better but more frequent pretraining. In contrast to regression, here we consider a classification problem and want to regard the DP learning approach.
As an improvement for models trained by contrastive learning to stealth them against backdoor attacks, Huan, Li et al. present a mechanism to decouple the training process [HLW+22]. For this, they train a model by the SimCLR approach and freeze the model after finishing. After that they train a fully connected layer in a first step supervised with all data and in a second step semi supervised by removing untrusted label. This approach is compared with DP-SGD and ShrinkPad to evaluate the improvement by defending against such Backdoor attacks. We follow a similar approach by training a SimCLR model, freeze the layers and then train one more layer supervised. But we do not try to enhance the privacy mechanism. We only focus on increasing the model accuracy by adding more knowledge to the base model.

3 Background

As discussed in chapter 2.2, SimCLR is a widely used technique to train machine learning models. In the following, we will present an overview about the code and libraries, we build on. Modifying the code to meet our requirements is shown in chapter 5.

3.1 SimCLR

Based on the work of Chen, Kornblith, et al. [CKNH20] the Google research team provide an environment to pretrain, fine-tune and evaluate models based on a ResNet architecture for SimCLR [FSC20]. The solution is written in python and uses Tensorflow as a basis for calculating and the training itself. The code is split up in three main parts.
At first, there is the run.py, the main function of this implementation. Here, the authors provide functionalities for the training and evaluation. In addition, there is a mechanism for saving and loading checkpoints, that can be used to restore a previous state in case of an error or to build on existing checkpoints. In this part of the program, the authors implement all necessary steps to perform the training, such as gradient and loss calculation, themselves. At last they initialize everything we need to perform the training, like the dataset or the model, by calling other functions.
One of these functionalities is provided by the data.py. In this part of the application, the dataset is initialized, and the images will be preprocessed. For preprocessing, the data_util.py is presented. This function takes a picture and returns it in a specific form, so after that the picture has the right size and is clipped to values between zero and one. In the training case the picture transformation will be performed by using recolouring, flipping and more. The labels will be one-hot encode, which means that they are transformed from an integer to an array. Its size then equals the number of classes. The class will be represented as a one at the right index, so in a two class problem zero will be encoded as [1,0] and one as [0,1]. After preprocessing all pictures, the batched dataset will be returned.
In another step, the model has to be build and the optimizer must be specified. For this, the authors present the model.py. They use a ResNet model as basis and build on the projection head, which is needed for pretraining, respectively the supervised head, which is necessary for the fine-tuning and the evaluation. The ResNet architecture is provided by the resnet.py, which requires the depth we want to use. Relu is used as the activation

function. In addition to the optimizers Stochastic Gradient Decent and Adam, the authors also show the Lars optimizer [YGG17] which can be used for speeding up the training by using larger batches. The objective.py and metrics.py are used for loss calculation and the metrics update.
Since the work is originally implemented for Tensorflow version one and later upgraded to Tensorflow two, we will focus on the later release.

3.2 Tensorflow DP-SGD

Privacy safe learning is an increasingly widespread topic. As we want to use DP-SGD as an optimizer in our implementation, we can refer to the implementation by Tensorflow. They provide a special library called Tensorflow Privacy [ACP19], implementing different private preserving mechanisms, also DP-SGD.
The DPKerasSGDoptimiser need different parameters we have discussed in 2.5, like the L2 norm clip, the noise multiplier or the learning rate. After this, we can use DP-SGD as an optimizer for our SIMCLR training implementation and perform a DP training.
Because at the end of the training we also want to know what security guarantees our system can ensure, we also can use the function compute_dp_sgd_privacy, implemented by Tensorflow Privacy. This will tell us, which ϵ we can reach for a given δ.

3.3 T-distributed Stochastic Neighbour Embedding

T-distributed Stochastic Neighbour Embedding (TSNE) is a tool which we want to use for visualization. It helps us to represent high-dimensional data in low-dimensional spaces. For this, TSNE follows a two-step algorithm. At first, each pair of points in the high dimensional space gets a probability, which is high when the points are similar and low when they are different. After that, a similar probability distribution over the points is mapped in the low dimensional space. Meanwhile, the algorithm tries to minimize the Kullback-Leibler divergence, between the probabilities in relation to the position of the points in low-dimensional space. The Kullback-Leibler divergence is a metric to calculate the equality of two discrete probability distributions defined on the same probability space X. The result of this is that similar points are represented nearby, while different points are far away from each other.
For our applications, we want to use the implementation provided by sklearn [1]. Since the cost function of TSNE is not convex, different initializations lead to different representations.

[1] TSNE provided by sklearn: `https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html`

4 Problem Statement and Challenges

After introducing the main concepts and definitions, we need to better understand this work, we now want to present the problem we want to face, describe shortly the approach which we want to use to solve it and discuss some challenges that could occur during the work.

4.1 Problem Statement

Machine Learning is a very topical issue. As we have seen in 2.3, deeper networks do not necessarily lead to better results. Even continuous training does not always lead to an improvement in a standard learning process, and we reach the point where the training progress stagnates or even declines. This issue mainly relates to conventional, mostly supervised-based methods. In the case of the contrastive learning approach, we can achieve through multiple trainings that many more different positive and negative examples are created, and this could increase the accuracy of our model. So we need to understand the impact of further training to the model itself. In other words, does further pretraining help to increase the accuracy of our solution?
That is the main problem we want to investigate in the following chapters and the work in general. The approach that we want to consider here uses the combination of knowledge by using different datasets to improve our model. This strategy is beneficial when we have many small datasets, on which we want to train one large, cohesive model. But even detached from this approach, the combination of knowledge can help to develop complex models that can handle difficult tasks well.
In many areas, it may be necessary to protect the privacy of the data and prevent any leaks. For this, the idea of Differential Privacy is necessary to harden our solution against privacy leaking problems and provide a secure and reliable system. As we have seen before, one of the most important resources of our digital age is data, and we do not necessarily want everyone to know them. Especially in the field of medicine or research in general, people and institutions have a great interest in keeping results secret. In contrast, there is also data that is accessible to everyone, such as images of public figures, the general environment and much more. This creates a division into private and public datasets, some secret, and worthy of protection, others publicly accessible to everyone.

4.2 Approach Overview

In order to achieve this goal, we want to present a pipeline that enables us to create a reliable and robust solution based on a wide variety of models. For this, there will be a two-step experiment, consisting of a contrastive learning pretraining and a supervised fine-tuning part.
The first part should serve to create a reliable model, which can achieve good results for a given data set. This can be done based on different approaches, on an existing model or from scratch. Therefore, we want to use this basic model and train with the contrastive learning approach SimCLR and a public data set. The goal of this step is to create a model, which can also deliver good results for unknown data.
The second part is intended to use the already pretrained model and to fine-tune it for an unknown task on an unknown data set. We want to make sure that we carry out this training differential private in order to protect the sensitivity of the data and thus also ensure its use in security-relevant areas. At the end, we can measure the accuracy for the new task to evaluate the quality of the model itself.
The goal of this work is to develop, implement and evaluate this pipeline. For this, we want to use different datasets and base models to get a high variance in our results. The details of the pipeline and the implementation we are going to use is presented in chapter 5. The overarching goal is to examine how the fine-tuning of pre-trained models affects the performance of DP-SGD.

4.3 Challenges

In the last part of this chapter, we will discuss some problems and challenges, that may arise during work on the pipeline just presented.
As we have seen, our work will be based on the code provided by the Google research team and refers to the work of Chen, Kornblith, et al. [CKNH20]. Since then, the code has been grown and updated to Tensorflow version two. So we have to understand, how the authors realize the calculations and how we can access the functionalities we need. This could be quite difficult, since the code is only very rudimentary commented. So we have to invest a lot of time to clearly understand the implementation.
This is necessary, because we want to implement DP-SGD as an optimizer and perform the training, respectively the evaluation, on custom datasets. Since the authors implement the training steps and gradient calculation themselves, there is a conflict with the DP-SGD implementation by Tensorflow. This is because it is necessary to carry out the gradient calculation itself to ensure that the computation is done correctly and in a completely differential, private manner. In the implementation, these calculations are performed with-

out a focus on DP. So we have to find an efficient way to implement mechanisms to the training, that allows us to use DP-SGD.
Additionally, we want to use custom datasets while the code is specialized for Tensorflow datasets. These includes building information, like image count, size and more. However, since this library only provides very few face datasets, we have to load in data by our own. For this, we have to change some calculations, find a way to include and preprocess the images, and it has to be ensured, that everything is read in correctly.
Next to the challenges that can be attributed to the use of the code, there are also problems that can generally arise in the machine learning context.
At first, there could be an issue with the dataset we use for training. If we just show our model a specific type of images, like black hair people, it could be difficult for our model to generalize, if it receives a picture of a blond or brown hair person. So we have to ensure, that we provide a high diversity in our data, to ensure a good training of our model. A SimCLR specific difficulty in this context could be the transformations we use to build the positive pairs. So it could be possible, that a colour transformation is not beneficial, if we want to train a model to analyse the hair colour. Therefore a big challenge could be to find a good diversity in our dataset and compare different characteristics.
There could also be the problem of over-fitting. So it is possible, that our model forgets how to generalize and merely memorizes what the correct answer is for a specific dataset. This can cause the progress of previous checkpoints to be lost and the entire system to deteriorate. This could be an issue, as we want to carry on the pretraining for existing models to increase the accuracy by a higher diversity of input data.
We also want to take a look at the number of iterations we want to use for pretraining and fine-tuning. Based on the code we want to use and which is presented in section 3.1, the standard number of iterations for pretraining should be 1000. Because this is a high amount of resources and time we need to invest, we have to take a look, about how different models perform after various times.
At last, we will focus on the parameters for DP-SGD. As we have already seen, the choice of these plays a major role in the security guarantees that our model can end up achieving. However, there is a connection between the influence of our parameters on the accuracy of the model. If we choose our parameters too strong, the influence of each data point becomes very small. This can lead to good values in the safety guarantees, but at the same time can severely affect the performance of our model, since the model does not learn hard enough. On the contrary, if we select the parameters too low, which would weaken the influence of the protective mechanisms, which may allow us to achieve better results, but also not ensure sufficient security of our data points. So it is very important that we achieve a good and accurate choice of these parameters.

5 Approach

In this part of the work, we want to introduce the approaches and methods we use to reach the previously discussed goal. After that, the implementation details will be presented to show our code changes. The experiments and the discussion of these will be continued in chapter 6 and 7.

5.1 Method

Before we get into detail about the data pipeline we want to use, remember that the goal is to train a model which can guarantee reliable results on sensitive data. At the same time, this model is intended to protect the sensitivity of this data. For this, we had shortly introduced a two-step experiment which can be used to train such a model and perform a differential-private fine-tuning after that, to improve the results for a given, sensitive dataset. In this section, we want to fully present this mechanism and discuss all details about the data we provide and how we build up our final solution. You can get a rough overview of the system in Figure 5.1. Generally, the idea is that we split up the approach into two halves.
On the one hand, there will be the pretraining which uses public data. In this part we want to achieve a model, based on the ResNet50 architecture, which can perform a given task reliable. On the other hand, we want to perform a privacy-preserving fine-tuning on this given model to ensure also good results for another task on an unknown and sensitive dataset. For this, we just want to train one linear layer privacy preserving. The results we get from this are based on the quality of the underlying model.

5.1.1 Pretraining

As we have seen, pretraining means the training of the whole network and usually refers to the first training of a model on a given dataset and task. The goal is, that we can later on use the weights of this solution to train other models. The hope is, that this will then have a head start over a new, untrained model because it can fall back on what has already been learned. We want to use the contrastive learning approach, SimCLR to realize such a learning mechanism. As we have seen in section 2.2, contrastive learning approaches do not need specified labels for training, since they create artificial labels by creating positive and negative pairs. This will be very helpful, since our custom datasets do not provide a

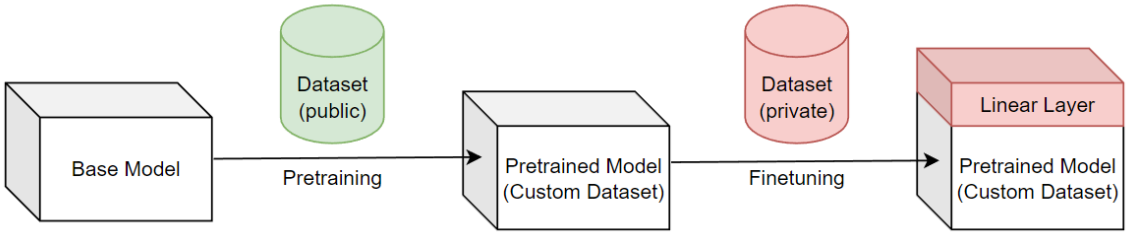


Figure 5.1: Overview of the two-step approach we want to use, consisting of pretraining and fine-tuning. Green represents the elements containing public data and do not need to be protected. All red parts have come into contact with sensitive data. The linear layer is trained supervised based on the embeddings of the pretrained model and the original labels.

uniform format for the labels. Thus we can use one of our datasets, we present in section 6.2, to perform such a training on our models. We just got the limitation, that we do not want to use one dataset in both steps of our approach. Notice, that the presented data is collected from the internet and thus publicly accessible for everyone, including potential attackers. So we do not need any mechanisms to protect the data at this point. After the pretraining has been carried out, the dataset can be used to validate the quality of the model for a given task based on this data, or it can get directly discarded. The technical way to use a custom dataset will be presented in 5.2. Until now, we just assume, that we find a way to correctly load a custom dataset. As we have discussed before, we want to use such datasets because we want a high variance in the training data. Additionally, we can modify them more easily than, for example, Tensorflow datasets by adding or omitting images.

In summary, we can say that we currently have a public data set that does not contain any information that is worthy of protection. We now want to use this dataset to train our model. For this, we can use two different attempts, we can train from scratch or on an already existing system.

From scratch means, that we can not fall back on existing weights. So we have to train from the beginning. A problem that normally occurs with this approach is, that we have to find good features which can be learned well and without prior knowledge. With SimCLR we do not have this issue directly, since the labels are created artificially, as we have seen before. But the creation of such pairs could be very difficult, so we have to ensure that we use a large batch size to generate as many negative pairs as possible. This approach should help us to evaluate the impact, if the model just had learned details about human faces.

In contrast to this, we want to use already pretrained models. For this, we can train models on Cifar10, Imagenet or similar datasets which provide images of animals, things of everyday life and much more, but not especially humans. We want to take a look about how all these different representations help our model to learn specified features of the human face. One thing that could happen is, that the training mechanism overwrites all the existing weights if they are useless. This is because we want to adjust the whole model and the training process weights the things that lead to a large improvement in the system. In our approach, we will use Cifar10, since there are only subsets of Imagenet available but not the complete dataset.
We want to use the SimCLR implementation [FSC20] by the Google research team to perform the pretraining. At the end of the training we are able to build the saved model with the given weights or just use the checkpoints to load the weights into a fresh model. After that, we want to perform the fine-tuning and our evaluation for an unknown dataset and task.

5.1.2 Fine-tuning

In the second step of our pipeline, we want to perform a fine-tuning on the given, pretrained model. It is possible to adapt the complete model. In our case, we freeze all layers we receive and add one linear layer on top. This will be trained supervised with the embeddings of the pretrained model and the original labels provided by the dataset itself. We do this because we want to use as much knowledge as possible from our pre-training. Accordingly, we do not want the fine-tuning process to change the weights of the underlying model, since this means changing the pre-training.
We want to deal with sensitive data given to the model and implement an algorithm, like DP-SGD, to secure our system and avoid the leakage of data afterwards. So in this part of the approach, we want to focus on the training with such data. For this, it is very important that we use the sensitive dataset only during the privacy-preserving fine-tuning, since we do not have any mechanism to protect them before this step. In this work, we will use CelebA mainly for fine-tuning. In this case, we just assume that this data is sensitive, although the data is publicly accessible and therefore not actually worthy of protection. The motivation behind this is that we want to investigate whether, for any model, DP-SGD can achieve good results in the finetuning case. This can then be generalized to any, possibly sensitive data sets. During this part of the approach, we want to measure the training and validation accuracy with given metrics for evaluation. So at the end of the finetuning we can determine the quality of our model and the training itself.
The goal was to train a model which can achieve good accuracy results on a sensitive dataset and evaluate the performance of DP-SGD on the given model. At the same time

our requirement for (ϵ,δ)-Differential Privacy has to be satisfied. Since DP-SGD is very delicate to the given parameters, it could be difficult to choose them in such a way that we fulfill the desired properties. For this we have to adjust the given training mechanisms as we have seen in section 4.3. This changes will be discussed in the next section.

5.2 Implementation Details

In section 4.3, we discussed two challenges where we need to adjust the code. On the one hand we have to deal with the implementation of DP-SGD, on the other hand there is the import of custom datasets. In this part of the work, we will show how we implemented the solution in the code, so that it meets our requirements.

5.2.1 DP-SGD

As we have seen, we want to use Differential Private Stochastic Gradient Decent as an optimizer. So far, SGD, Adam, and the Lars optimizer have been implemented as standard in addition to this, but we want to ensure the sensitivity of the data during the fine-tuning of the model. For this, we need an optimizer that can perform differential private preserving training, like DP-SGD.
Since we had to realize that fine-tuning is still implemented in the Tensorflow two implementation of the training mechanism, but another way is presented in GitHub to fine-tune models, we can only partially use the previous code. Therefore, we use the provided google colab[2] to fine-tune models. In this part of the code many functionalities equal the original code, like the image preprocessing. But there are some changes on the model architecture, since this code is not designed to train the whole model further. In this case we can not load checkpoints to carry out the further training but have to provide ready-made models. However, we can avoid this by initializing the models according to the usual structure and then loading the given checkpoints. This allows us to efficiently load larger models and run a variety of experiments at different points in the training.
After that, we now have to find an efficient way to introduce DP-SGD as an optimizer. For this, we will use the Tensorflow privacy implementation, as presented in section 3.2. DP-SGD wants to calculate the Gradients by itself. So if we want to use this optimizer for fine-tuning our model, we can not fall back on the code provided by the Google research team. This is the case because they carry out all the calculations themselves, including those for the gradients, and adapt the model themselves. Since DP-SGD does not allow that, we have to find another way to do the fine-tuning. Because the fine-tuning corre-

[2]Finetuning for the Tensorflow two implementation of SimCLR: `https://github.com/google-research/simclr/blob/master/colabs/finetuning.ipynb`

sponds to the supervised training of one layer. For this, we can use the fit function which is defined on Tensorflow models. This function can be used to train a model on a dataset for a specified number of epochs and for a given loss function. Since fit use the call method of the model itself, we can not use the normal model structure provided by the implementation because fit does not know how to handle this. We want to use the embeddings of the pretrained model to train our linear head. So the approach is, that we define our model as a system consisting of a linear layer and fit it with the dataset including the embedding of the pretrained model and the label. For this, we have to load our image dataset normally and perform the image preprocessing. After that we perform a new transformation where we replace the image by the output of the model for the given picture. At this point, we are able to train our linear layer based on the embeddings of the pretrained model. For the training step itself, we will use the Categorical Cross entropy and the Categorical Accuracy for this since they are common for datasets with one_hot encoded label.
Now we can start to fine-tune our model by using the fit function. This needs the dataset itself and the epochs we want to perform the training on. This function returns the training history, which we can use for visualization. But if we want to compare the training and the evaluation accuracy, we need to provide a train and validation dataset. Because there is no implementation in the current application, we require a solution made by our self. What we want to realize is a function, which builds and preprocess the dataset, splits up and returns both parts. Since we now use the different implementation of the fine-tuning mechanism, we no longer depend on the function that returns us a distributed dataset that we can not easily split. So we can just follow the approach by the take and skip function defined on Tensorflow datasets. This allows us to define how many entries we want to take respectively skip from a given dataset. So if we take 80% of our data and skip the same size, then we successfully performed an 80/20 split, which is a common size for splitting the data. In order to avoid a possible imbalance in the split and to avoid potentially bad splits, we shuffle the dataset before that. If the program is run several times, this leads to different divisions and thus eliminates the problem.
We want to evaluate which security guarantees our model fulfils. For this, we can use the functionalities provided by Tensorflow Privacy. Which gives us the ϵ value that our model can fulfil for a given number of training examples, the batch size, the noise multiplier and a δ chosen by us.
So at the end of this part, we are able to use DP-SGD as an optimizer for differential private training. Additionally, we are able to calculate the security guarantees our training can ensure.

5.2.2 Custom Datasets

After that, we have to discuss the implementation for custom datasets. As we have seen, we can not just fall back on Tensorflow datasets (tfds). So we need to find an efficient way to read in custom data. For this, we have to start collecting the different sets from the internet and take a look which format we can receive. All of them are available as a collection of images, so we just have to find a way to read in images as a Tensorflow dataset. For this, the function image_dataset_from_directory, provided by Tensorflow, can be used. This allows us to define a directory of images, and the function will read in the data as a dataset. Additionally, we can give an array of labels, if we can not provide a structure or a label list in the directory itself to automatically generate them.
Since our custom dataset can not provide a tfds builder, we have to remove this part out of the code. One problem that arises from this is that the original implementation does a lot of calculations based on the builder, like the number of train examples or the number of classes. Since we load the data out of a directory, we can do the most of the calculations above the number of elements in the directory, but there is no efficient way to count the number of classes. One possible way is to compute the labels and take the length of the set of these.
Another Problem occurs during the loading of the picture with the associated label. As we do not have a structure, that can be used to automatically generate the labels, we have to find an own way to read in the labels. In the case of the CACD dataset, the labels are part of the image title. If we now want to list them with the operating system function listdir, provided by python. We had to realize that the titles are not read in alphabetic order. Consequently we have to sort them before building our dataset.
The labels will be one-hot encoded. So we have to ensure, that all our labels correspond to a value from zero to the number of classes. In the case of the CACD dataset, where we got the age as a number between 14 and 62, we have to subtract 14 from each class. This is necessary because if we do not do this, the 14 is not the first class encoded and the higher ages are not completely represented in the encoded labels. In the case of a character string as a label, we also have to translate this into a number representation so that it can be displayed adequately. If we want to decode them, for example for visualization, we just have to take the index of the one in the array, as there is no function to reverse the encoded data. This can be realized by the NumPy function 'where', which returns an array with the corresponding indices. Since there is always only one label per image, the result array has sized one.
After implementing this mechanism, we have to ensure, that everything was loaded and processed correctly. As you can see in figure 5.2, all images are read in correctly and in the same size. In this case, we use the hair colour of the people as a label to demonstrate

Figure 5.2: Images of the CelebA dataset loaded custom as a tf dataset.
Zero equals blond, two equals black and four equals brown hair.

the correct assignment. For this, we transform the different hair colours into a number representation, so they can be transformed in the one-hot coding later on.

At the end of this section we will be able to train differential private using DP-SGD. Furthermore, we can load data sets whose contents we have as images in a directory. This allows us to use a variety of datasets to conduct our training and study the impact on DP-SGD.

6 Experiments

In this work, we will focus on a two-step experiment. In a first step we want to train a model, based on the ResNet-50 architecture, and later on we will fine-tune this model for a specific task. We want to investigate, how different pre-trained models will perform. The goal is to pre-train a system that can ensure a high accuracy on a provided task for an unknown data set after DP fine-tuning. In this chapter, we will present the datasets we used and the experiments we performed. Based on this we will present the results we received. An interpretation of these is shown in chapter 7.

6.1 Technical Overview

For the experiments, we will work with the NVIDIA DGX A100 [3]. This allows us to perform the training on larger batch sizes than on usual graphic cards.
A general overview of the usual training parameters are shown in table 6.1. Notice that we will use the Lars optimizer for evaluating our model. This will be presented in section 6.3. Since there are more specific parameters we need for DP-SGD in general, we will discuss them in the specific section. All parameters for SimCLR training equal, unless otherwise stated, the information provided by the Google research team. [4].

Property	pre-training	DP-SGD fine-tuning	Lars fine-tuning
Batch Size	512, 1024	512	512
Learning Rate	1	0.25	0.1
Image Size	32x32, 224x224	32x32, 224x224	32x32, 224x224
ResNet depth	50	(50)*	(50)*
Epochs	1000	100	100

Table 6.1: Overview about the training parameters we want to use in our experiments. Specific parameters will be discussed in the sections. The Lars fine-tuning will be used for evaluation. *The ResNet depth in the fine-tuning case describes the depth of the ResNet of the underlying model.

[3]Technical Information about the DGX A100: `https://www.nvidia.com/en-us/data-center/dgx-a100/`
[4]Parameters for pre-training: https://github.com/google-research/simclr/tree/master/tf2#pretraining

6.2 Experimental Setup

To train a model and perform an evaluation later on, we will shortly present the four datasets we want to use. Since the goal is to train a model that is as generalized as possible, we want to have the greatest possible variance in the training data.

6.2.1 CelebFaces Attributes Dataset

The CelebFaces Attributes Dataset (CelebA) [LLWT15] is an extensive dataset with 202.599 face images from 10.177 identities. For each image, the authors provide 40 binary attributes, like bags under eyes or smiling. We will use this data mainly for fine-tuning and evaluation, caused by the high amount of features provided with the image data.

6.2.2 Cross-Age Celebrity Dataset

In the publication of Bor-Chun Chen, Chu-Song Chen and Winston Hsu about the Cross-Age Reference Coding for Age-Invariant Face Recognition and Retrieval, the authors provide a system to encode the low level feature of a face image. To realize this, they present a celebrity dataset, called Cross-Age Celebrity Dataset (CACD) [CCH14]. It contains 163.446 images of 2000 celebrities with an age between 14 and 62. It is also known as CACD2000.

6.2.3 IMDB-WIKI Dataset

As a mixture between the IMDb dataset and images from Wikipedia, IMDB-Wiki [RTG18] contain 523,051 pictures in total. The authors chose the persons after the list of the most popular 100,000 actors, listed by the IMDb website and collected pictures with the same metadata, like date of birth, name or gender, from Wikipedia. The authors provide information about the picture by assuming the images show the face of the actor himself. The identity, the year of birth and the year the photo was taken are given.

6.2.4 Labeled Faces in the Wild Dataset

The Labeled Faces in the Wild dataset [HRBLM07] includes 13233 images from 5749 people collected from the internet and provided for studying the challenge of unconstrained face recognition. For this, the authors supply the identities of the people depicted in the pictures.

6.3 Experiments

In order to be able to ensure a versatile evaluation of our pipeline, we want to pre-train different models on different data sets. We then want to use all models to carry out the same tasks over CelebA. An overview of the models that we train and the combination of the different datasets is shown in figure 6.1. Notice, that each path uses the data set only once and the evaluation always takes place with an unknown data set (CelebA). Only in the case of the CelebA model itself do we want to use the same dataset twice to evaluate whether our model architecture is sufficient to achieve good results for CelebA using contrastive learning. Furthermore, lower models always build on their predecessors, i.e. load their checkpoint and continue training with it. As we discussed earlier, this is because we want to combine knowledge from models on different datasets to create a model that can accurately solve difficult tasks.

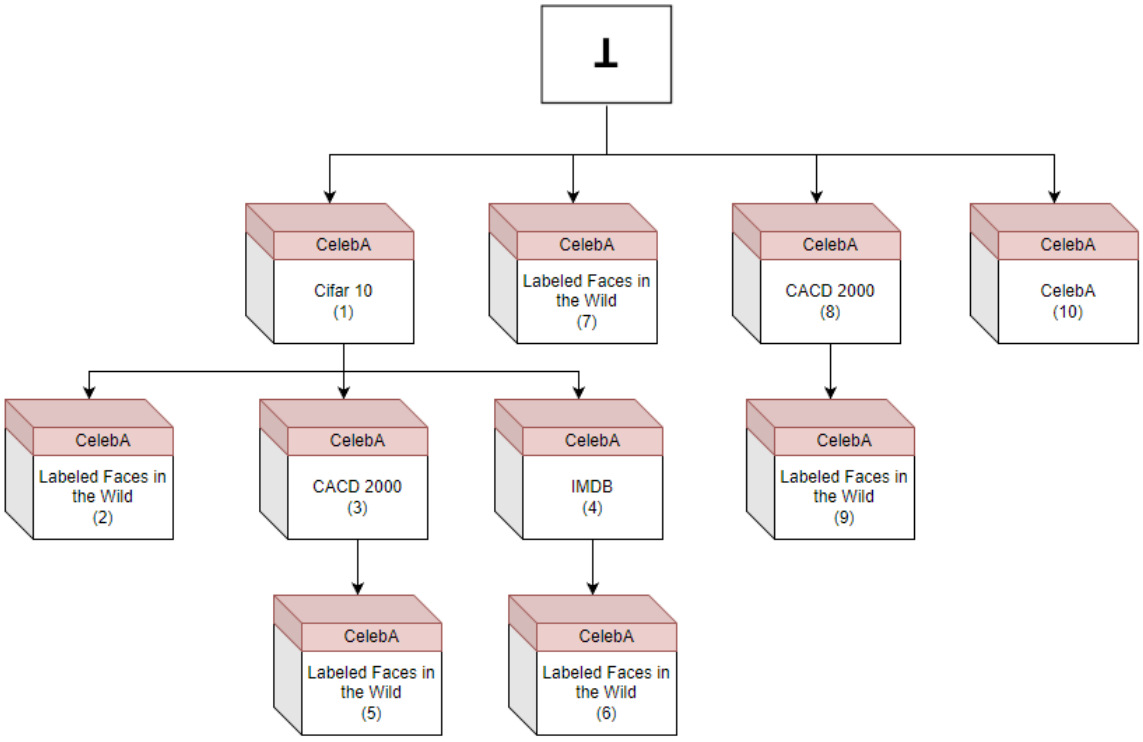


Figure 6.1: Overview of the different models that we want to pre-train and then evaluate. Subsequent models build on the last checkpoints of the predecessors. The first level of models is trained by scratch and thus has no predecessor. Each model is given its own id, making it easier for us to address them.

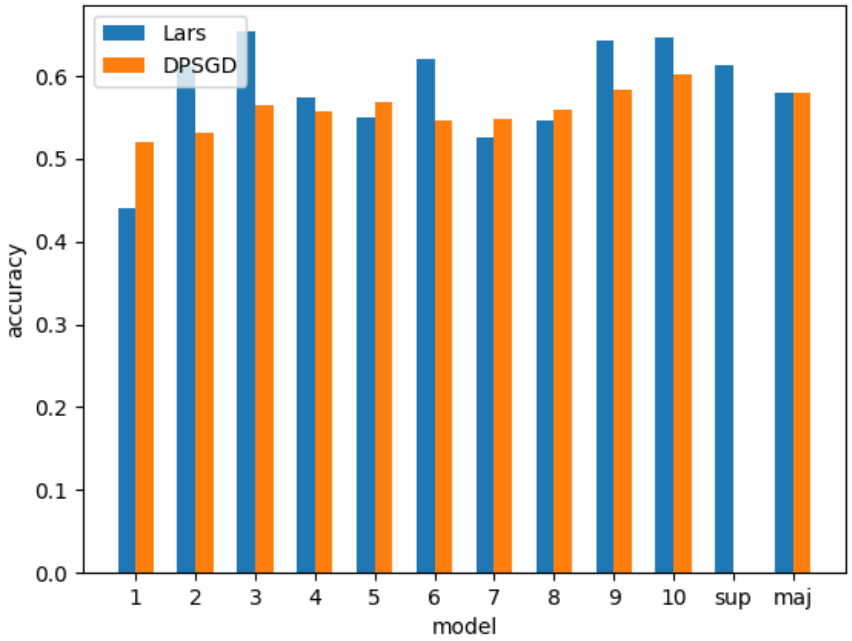
Differential private training mechanisms try to reduce the impact of single data points to the model. In the case of DP-SGD, this is realized by clipping the gradients and adding noise to them. These modifications generally lead to DP learning methods achieving poorer results than methods without any data protection. To avoid misunderstandings and to get a measure of how further pre-training affects the quality of our model in general, we want to consider two different approaches to evaluate our system. On the one hand, we will perform the fine-tuning with DP-SGD to ensure secure training with sensitive data. On the other hand, we will use Lars as the optimizer to get a feel for how accurate our model can be in general. This step only serves to evaluate our results for further pre-training and is not part of the final solution, since no privacy protection mechanisms are used. For fine-tuning we want to use the CelebA dataset. This is a publicly available data set. For our purposes, we want to assume that the data is sensitive and therefore worthy of protection. However, we can also use this data set to achieve a good baseline by briefly ignoring the protective mechanisms.

6.4 Experimental Results

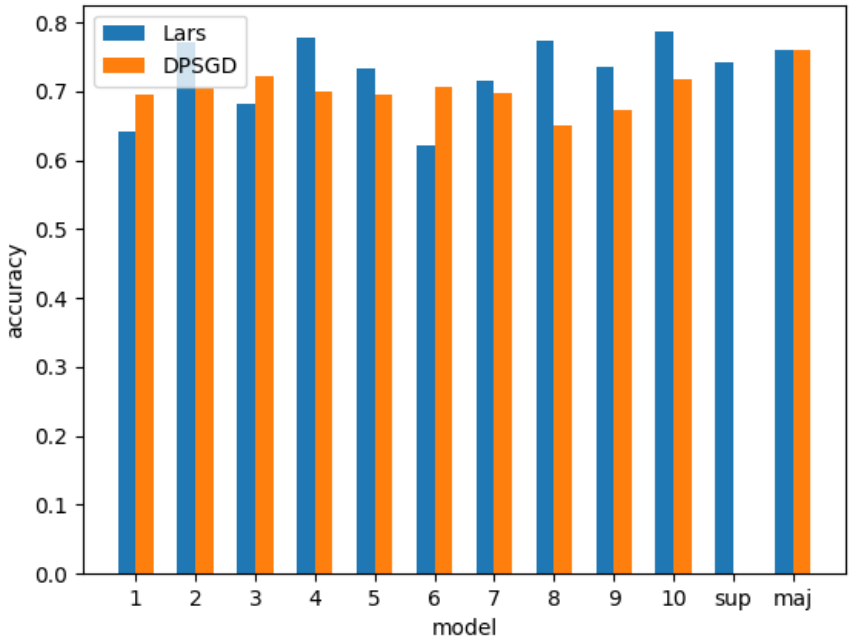
In this section, we want to present the results of our models that we introduced in figure 6.1. To do this, we want to split the evaluation into two parts, on the one hand the pure accuracy of the models and on the other hand the DP guarantees that we can ensure.

6.4.1 Model Accuracy

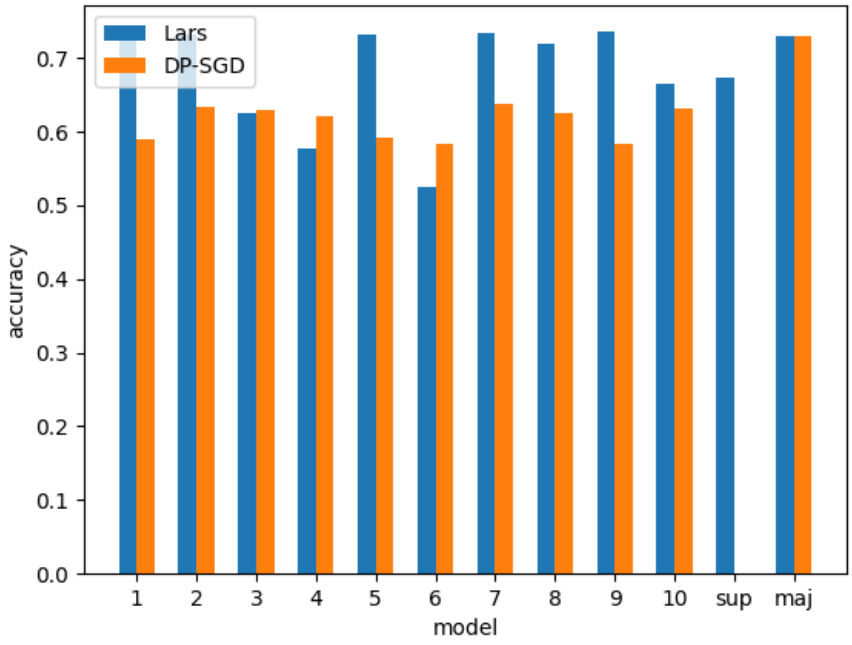
To check how well our models perform for different tasks, we want to present three different features. For this, we want to let the model first decide whether the person on the image is a man. The second task is for our model to decide whether the person has black hair. Finally, it should be decided whether the person has arched eyebrows. Notice, that the experiments are structured in such a way that they go from quite obvious features to more and more detail. Since all properties of the images for CelebA are always binary problems, we always challenge our model whether the property is satisfied or not. As we discussed before, we always want to compare our DP results with Lars as well, i.e. a mechanism that cannot ensure differential privacy. The goal is to determine how well our model performs in general without reducing the impact of the training points.
In addition to the results, we want to indicate what accuracy a model would achieve that simply always guesses the larger of the two classes. Such a model achieves the accuracy of the percentage of this class in the data set. Furthermore, we show what a model with the same architecture (ResNet50 + one linear layer) can learn supervised for the given task. These approaches are mainly discussed in the interpretation during chapter 7.



a) Results for the male decision problem.



b) Results for the black hair decision problem.



c) Results for the arched eyebrows decision problem.

Figure 6.2: Accuracy results for the given models to decide whether a man is depicted or not (a), to decide whether the person has black hair or not (b) and to decide whether the person has arched eyebrows or not (c). Sup stands for the results that we can achieve through supervised training. Maj for the accuracy we receive for a model, that just always guesses the larger of the two classes and thus achieves the accuracy of the distribution of the largest class in the dataset. Running over 100 epochs, for a $\delta = 10^{-}9$ with noise multiplier = 2.45, we get $\epsilon = 0.982$.

The results of the evaluation are shown in figure 6.2.
As we can see, all models achieve only marginal improvements compared to a model that decides based on the size of the respective classes in the data set. It can even happen that our models deliver worse results. This suggests that the training of our models is insufficient to reliably learn and discriminate the features of CelebA.
We can continue to observe that in most cases the accuracy of the Lars optimizer is better than that of DP-SGD. This was generally to be expected, since DP-SGD uses methods designed to minimize the impact of individual data points. One effect of this may be that it reduces the impact of training. Accordingly, non-private mechanisms, such as Lars, usually achieve better results.
Furthermore, notice that all models perform approximately the same for the given tasks. We can also see that the results are independent of whether a model has already been pre-trained or had to try to extract features from scratch. Additionally, it is also not relevant whether a facial data set was used, since the cifar10 model also delivers similar results to models trained with facial data.
It is also noticeable that the model trained for CelebA also delivers average results. This may mean that our training is insufficient to reliably classify CelebA, or that our model architecture may not be extensive enough.
In summary, it can be stated that our models cannot achieve reliable results for the tasks set. There can be various reasons for this, which we will discuss and analyse in detail in chapter 7.

6.4.2 Differential Privacy Guarantees

As we have already seen, in the DP-SGD case we can also calculate the safety guarantees of our training. These are very important, as they give us a feeling of whether our training has been carried out differential private, or whether attacks on the data and thus a potential disclosure of information are possible. Since we use the implementation of DP-SGD provided by the Google research team, we can also use their analysis of the security guarantees as described above. The precise calculation of the values is implemented using various functions, which we will not look at in more detail. These can be found in the implementation itself [5].
In this part, we just want to look at the guarantees we can achieve for the CelebA dataset because we use it for fine-tuning with DP-SGD. We know that CelebA has 202,599 entries, and we want to use 80 percent of them for training, i.e. 162,079 data points. These are

[5] Calculation of ϵ based on the Google research implementation: `https://github.com/google/differential-privacy/blob/main/python/dp_accounting/rdp/rdp_privacy_accountant.py`

Number of Epochs	Number of Entries	Noise Multiplier	Batch size	δ	ϵ
10	162079	1.3	256	10^{-9}	0.978
20	162079	1.4	256	10^{-9}	0.981
50	162079	1.8	256	10^{-9}	0.987
100	162079	2.45	256	10^{-9}	0.982

Table 6.2: Overview about the differential private guarantees with various number of epochs. These values are calculated for the CelebA dataset with 202,599 entries where 80 percent is used for training, i.e. 162,079 data points.

therefore worth protecting and must be taken into account in the calculation. As we have discussed in section 2.3, δ should be smaller than $\frac{1}{|D|}$ for a given dataset D. Therefore, we want to choose $\delta = 10^{-9}$. ϵ, on the other hand, should be picked to be smaller than one. It was shown that most attacks on DP training methods can only be successfully carried out with an $\epsilon > 1$ [RRL$^+$18].
A calculation for the guarantees we can ensure and the parameters we have to use for this is shown in table 6.2. Notice, that increasing the number of epochs means that we have to add significantly more noise to the gradients. This is necessary because running the training multiple times over the same data points will result in them having more of an impact on the overall model. This increases the risk that these can be identified. Accordingly, the noise simply has to be chosen higher in order to reduce the overall influence.

7 Evaluation

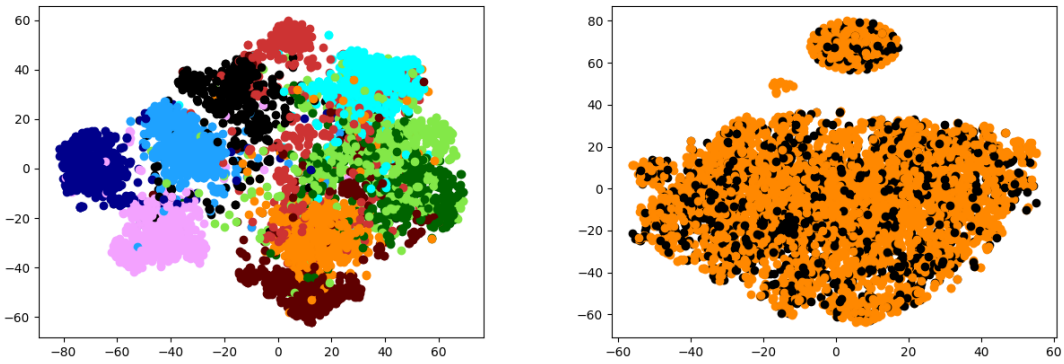
In this chapter, we finally want to interpret the results of our experiments shown in chapter 6. For this, we want to focus on different aspects that could explain our results. After that, we want to discuss the challenges, which we formulated at the beginning of this work.

7.1 Interpretation

As we have seen in section 6.4, with the models we have trained we do not achieve the accuracies that the distribution in the dataset exceeds. In this section, we want to discuss why this could be and interpret our results. To do this, we want to examine our previous models again and make additional modifications to our experiments in order to determine possible reasons for the performance.

7.1.1 TSNE

First, we want to check if we achieved that we can recognize clear separations in our data set as a result of contrastive learning. So let us observe whether the formation of positive and negative pairs has led to the formation of meaningful clusters. To do this, we want to display them using TSNE. We want to use the images from our dataset, iterate them through the model, and then display the results. Notice, that the results we receive from here are the input of our linear layer. Since these are high dimensional, TSNE has to do the mapping into a low dimensional space, to display the results in two dimensions. To better represent how the clusters look exactly, we want to colour each embedding with the corresponding colour, which depends on the class. Therefore, each class gets a colour, and we expect a good SimCLR model to have data of the same class closely spaced.
To illustrate this approach in more detail, the representation of the Cifar10 dataset on the Cifar10 model we use is shown in figure 7.1a. As we can see, there is clear clustering, disturbed only by a few outliers. The model is therefore able to pack images of the same class together and distinguish them from other classes.
However, if we apply this procedure to the CelebA model we have trained, we get the result shown in figure 7.1b. As you can see, the TSNE does not look sorted, which suggests that our model does not allow a clear separation of the clusters. As a basis for this image, we took the images from CelebA itself. Normally, it is to be expected that a model trained



(a) TSNE Cifar10

(b) TSNE CelebA

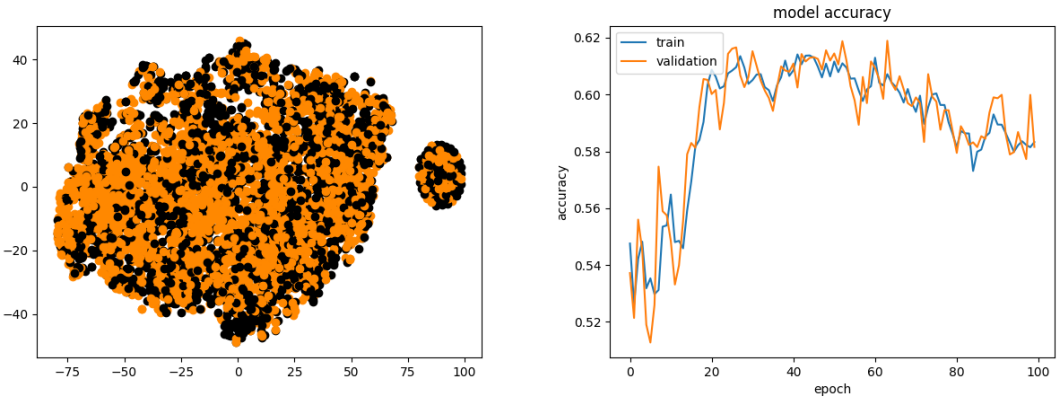
Figure 7.1: Depiction of the embeddings of the images in the models trained for the specific dataset. The evaluation of the 500 epochs Cifar10 model is shown on the left, the evaluation of the 1000 epochs CelebA model on the right. Both models were trained with SimCLR.

on the CelebA data set should be able to clearly separate it. So we need to investigate what causes these results to occur. What we can definitely state is that the pretraining that we used is not sufficient.

Furthermore, we can conclude that if the CelebA model is not able to produce accurate results on this data set, then the other models probably have little chance of success either. So we certainly have to adapt the pretraining process.

Foremost, we want to try whether adjusting the image size has an impact on the quality of the training. To do this, we increase it from 32x32 to 224x224. The motivation why we wanted to use 32x32 images was that this makes the training process more time-efficient because not such high-dimensional data is used. Furthermore, small images have the advantage that you can load more of them at the same time, so the batch size per training step can be increased significantly. As we have already discussed, it is important to have large batch sizes so that as many positive respectively negative examples as possible can be formed.

Unfortunately, due to the resources required to process 224x224 images, it is no longer possible to guarantee batch sizes of 1024 or 512, so we have to fall back on 256 images per batch. At the same time, this means that we need more training steps to train the same number of epochs. To check whether this adjustment brings an improvement, we train a model on CACD from scratch and an LFIW which is based on the CACD model. Due to time constraints, it is not possible to train 1000 epochs CACD because the data set is very large. Accordingly, we only pretrain 400 epochs and then use this checkpoint for



(a) TSNE CACD 224x224 images

(b) Training curve CACD 224x224 images

Figure 7.2: Depiction of the results for the modified model with image size 224x224. On the left side the embeddings of the images are shown, while on the right sight the accuracy of the model for the decision problem whether a man is displayed or not, evaluated with DP-SGD.

1000 epochs LFIW. Apart from the adjusted image and batch size, we continue to use the parameters that we presented in chapter 6.
The results of this approach are shown in figure 7.2. Here, we evaluate the new CACD model from scratch on the CelebA dataset. The results of the LFIW will not be discussed further here, as they are almost identical.
Notice, that we can receive accuracy results between 0.58 and 0.62. As we have seen in section 6.4, the scratch CACD model, which was trained with 32x32 images, achieves an accuracy of 0.56. So we can see that adjusting the size of the images also contributes to a marginal improvement in the model. However, clear clusters still cannot be separated, as shown in figure 7.2a. Therefore, simply adjusting the image size may not be enough, or we may have to extend the pretraining of the model in order to be able to achieve even better results.
The challenge of clear clustering can also have other causes. For example, it may be that the standard transformations, such as rotating or recolouring, that are used during the pretraining are not sufficient to train the model well for CelebA features. An adjustment of these may therefore be necessary. This approach is not considered further here.

7.1.2 Training Curve

Since we want to investigate in this work to what extent the pretraining of models affects the performance of DP-SGD, it makes sense to examine the training process of DP-SGD in detail. For this, we want to look at the following training curve of the CACD model for

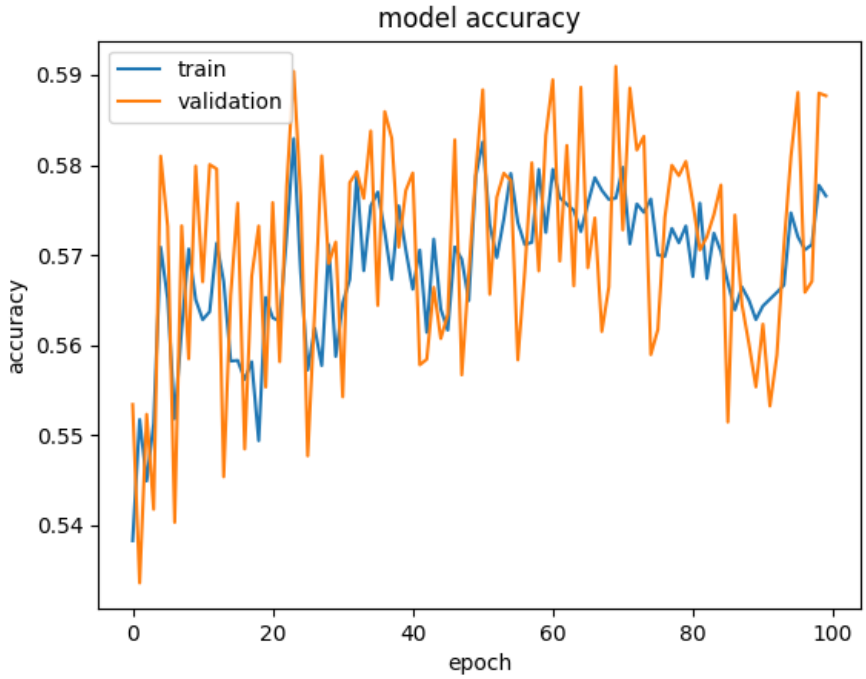


Figure 7.3: Display of the DP-SGD training curve on the scratch CACD2000 model for the decision problem whether a man is displayed or not. The evaluation was performed on the model trained with 32x32 images.

32x32 images. This is shown in figure 7.3.
As you can see, there is no consistent training line here, but spikes in accuracy up and down. While these are not huge, they can be an indication that our linear layer is trying to learn features. Remember, that our model architecture was structured in such a way that we take the pretrained model and want to train a linear layer on it using DP-SGD. Exactly this layer now tries to extract characteristics and to achieve a separation in the dataset. The deflections are a sign that such a separation is not possible and that the model is trying to find other separations again. As a result, we do not achieve consistent improvement or stagnation in our model, but such jumps.
This may be because due to the underlying, pretrained model not being able to achieve an unambiguous separation, as we discussed in the previous section. So it is also difficult for the linear head to perform a separation, since it has to build on the unsorted outputs of the pretrained model. This means that the linear head cannot contribute to delivering good results either, since it does not learn sufficiently on its own.

7.1.3 Evaluation with IMDB dataset

Another possibility that could explain why we are getting these results is that our architecture with a Resnet 50 and a linear layer on top is not sufficient to be able to learn CelebA features reliable. Therefore, we want to evaluate whether our models can deliver good results on a different data set.
For this, we want to use the IMDB dataset. Notice, that in addition to the identities of the people their date of birth and the year the photo was taken is provided. So we can calculate the age of the people in the photo and thus get the identities and the age as a label. We want to use a modified form of the dataset here, since the original data is highly unbalanced. Some classes are represented with less than a hundred photos, while the main age classes contain several thousand images. Therefore, in our modified variant, we only want to look at the classes with at least a thousand pictures and only take a thousand of them per class. So the data set then includes people between the ages of eleven and seventy, and thus 60,000 photos. In order to make the comparability to the CelebA dataset, we want to continue to consider a two-class problem. So if we select all persons up to 40 in one class and all others up to 70 in another, we get 30,000 pictures per class.
The evaluation of our experiments is shown in figure 7.4. Notice, here we took the same parameters for DP-SGD as for the CelebA data set, although that would not have been necessary. However, we want to ensure that the comparability of both experiments remains largely intact. Normally, due to the significantly larger number of data points in IMDB, a smaller noise multiplier would be possible to achieve the same security guarantees.
We can state that except the Cifar10 model, all models achieve at least the accuracy of 0.5. However, this is probably not due to the fact that the features were extracted well. Rather, the reason could be that the distribution of the data to class zero or one is exactly half. The results of our models can have various causes. On the one hand, our models may not be able to separate this dataset well either. On the other hand, it is possible that we need other training methods in order to be able to classify the age of people well. An approach to this was presented in the section about related work, and adapts the pretraining so that positive pairs are formed by people of the same age and not by image transformation [LGW+21].

7.1.4 Implementation without DP-SGD

As we have seen, DP-SGD uses gradient clipping and noise to reduce the impact of individual data points, thereby protecting sensitive data. We had also discussed that this can lead to the results in the accuracy of the model being reduced. Accordingly, we want to examine here what influence DP-SGD has on the training of the linear layer and thus

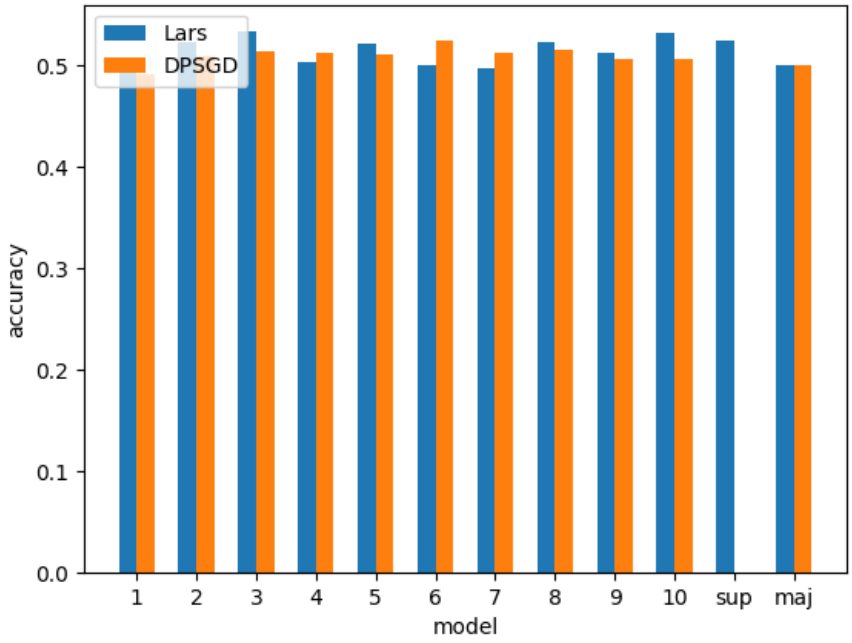


Figure 7.4: Accuracy results for the given models on the age decision problem for the IMDB dataset.

on the quality of our pipeline. For this, we can use our models and carry out the same experiments without paying attention to privacy preserving training.
As you may have noticed in Chapter 6, we use Lars as an optimizer for this. The results of our evaluation are plotted simultaneously with the results of DP-SGD for all experiments. Regarding figure 6.2 and figure 7.4, we can see that Lars performs only slightly better or even worse than DP-SGD, but these also do not exceed the value of the distribution. Therefore, the pretraining has to be improved in order to achieve good results for the model in general.

7.1.5 Supervised Learning on Resnet50

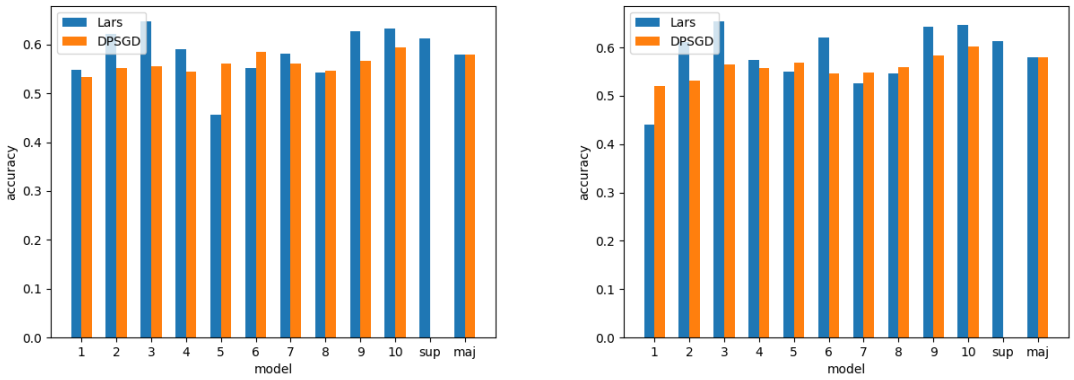
Last but not least, we want to check in another way whether our architecture is able to learn CelebA by leaving the contrastive learning approach and doing purely supervised training. For this we build a model consisting of the Resnet50 architecture, which is provided by the Google research team, and pack a linear layer on top of it. Notice, that this is the general structure of our pipeline, only now we do not use a pretrained model and

only carry out supervised training. Since we just want to check how well our model architecture is designed for CelebA, we want to take a non-DP approach to achieve the best possible results. For this, we can fall back on the Lars optimizer.
The results of this approach are also shown in figure 6.2 and figure 7.4. We can see that even the supervised results are not enough to get good results for the given tasks. This suggests that our model architecture may not be sufficient to reliably learn features of CelebA. This can either be because the one linear layer is not sufficient to extract unique features based on the Resnet model. However, it is also possible that a Resnet with 50 layers is not deep enough.

7.2 Discussion of the challenges

In the last part of this chapter, we want to summarize the challenges that we discussed in section 4.3. We want to take a quick look at what the expected difficulty was and what we implemented to solve the individual tasks.
First, we discussed that we want to use DP-SGD as an optimizer. This was necessary because in this work we want to investigate how the fine-tuning of pretrained models affects the performance of DP-SGD. The challenge was that in Google's implementation, all calculations for adapting the model were implemented in the custom training step. This applies in particular to the calculations of the gradients. But DP-SGD requires that these are carried out by the functionalities provided by the optimizer in order to be able to ensure that the training is really carried out differential private. Accordingly, the training had to be adjusted so that all adjustments to the model are performed using DP-SGD. Our implementation is shown in section 5.2.1. We implemented the training in such a way that we load our pretrained model as usual and preprocess the images in such a way that the data set then consists of the image embedding and label. After that, we only had to define a linear layer as a model and could then train it using the fit-function. This has enabled us to use DP-SGD.
As a second challenge affecting the code, we had motivated why we want to use custom datasets. The reason for this was that we wanted to use the combination of knowledge to train reliable models. Furthermore, we explained that with such an implementation, we can access numerous data sets that are easy to modify. With the introduction of such data sets, the problem was that we could no longer access many of the code's calculations, since these were mostly realized using Tensorflow datasets. So we had to find an efficient way to do this ourselves. We presented our implementation for this in section 5.2.2. The most important changes were that we can load images from a directory. These are then available to us as a Tensorflow dataset. However, since these are still not part of the Ten-

sorflow dataset library, they do not have a builder and therefore no meta information such as number of images or similar attributes. We have shown how these calculations can be performed using the directory structure. This allows us to load custom datasets correctly, which was necessary because it enabled us to carry out many experiments on the various data sets. This would not have been possible with the standard data sets.
Away from the code, we showed the challenge that machine learning and the DP-SGD approach can entail. One of them was the imbalance in the dataset. The problem here was that, especially in supervised training, such an imbalance can mean that our model can deliver good results for certain tasks, but can only solve other tasks poorly or not at all. Especially in the case of our fine-tuning, it was important that we prevent such an inequality. This challenge is also of great importance in the interpretation of our results, where we use a supervised approach as a comparison to our SimCLR models. The solution is that we look at relatively balanced attributes in our experiments. In the case of our fine-tuning, this is guaranteed by only using classes whose largest class is a maximum of 70 percent. In the example of supervised training in section 7.1.3, we even showed that we modify the data set to correct an imbalance. This is mainly possible because IMDB is a custom dataset. Such imbalances during pretraining are avoided from the outset by training many models on a wide variety of data sets. This gives us a high variance in our image data.
Another problem associated with using datasets can be overfitting. As we have already discussed, the issue here is that if the same data set is used too often, it is possible that the model will start to memorize it. This can result in the model no longer being able to generalize. We had also shown that this is rather less likely in the contrastive learning approach, since more frequent training leads to the formation of more positive and negative pairs. This can help improve the model. However, this concern can arise during fine-tuning or our interpretation experiments. Since we have examined what results we can achieve with CelebA on different models trained with CelebA, we have clear evidence that no overfitting has taken place. This is because otherwise near perfect results would have been achieved since the model already knows the answer to the questions. Nevertheless, it cannot be ruled out that after a certain number of training steps, our models perform better than they are currently doing.
To investigate whether the number of training steps makes a significant difference, we want to evaluate the results of our models at 500 and 1000 epochs of training. The Google research team suggested 1000 epochs for training. But since this is associated with a lot of time and resources, we wanted to reduce it. The results of this approach are shown in figure 7.5.



(a) Training accuracy after 500 Epochs

(b) Training accuracy after 1000 Epochs

Figure 7.5: Depiction of the accuracy results for the models after 500 and 1000 training epochs. Here, the problem of whether a man can be recognized in the pictures or not was considered.

As one can see, there is no significant difference between the models after 500 and 1000 epochs. Notice, that some models achieve even better results when they have been trained less. However, this can also be due to the fact that we have the spikes in accuracy, as we showed in section 7.1.2. After an adjustment of the pretraining, it can be the case that the number of iterations has a significant influence on the results of our models. In this case, it would have to be examined at what point the training stagnates.
Last but not least, let us take another look at the parameter selection of DP-SGD. We discussed that the parameters have a high impact on the accuracy of DP-SGD, as they determine how high the impact of each data point is. Our reasoning was that overfitting causes an unnecessary loss of information, while the sensitivity of the data cannot be guaranteed if the choice is too low. Based on the work of Md Atiqur Rahman, Tanzila Rahman et al. [RRL+18], we had seen that we should achieve an $\epsilon < 1$ so that common attacks on DP are no longer possible. Furthermore, we found that our algorithm should deliver better results than one that randomly throws out any data point. So $\delta < \frac{1}{|D|}$ should hold for a given dataset D. Based on this, we made our choice of parameters. This was presented in section 6.4.2. Since CelebA is the only dataset that we want to use by default for our DP-SGD fine-tuning, our choice of delta is sufficient. From this, we can now calculate our other parameters. We have shown that these are chosen exactly such that $\epsilon < 1$ holds. In summary, we can say that we have selected the parameters for DP-SGD in such a way that they achieve the desired security goals. However, it is important to say that each data set has its own size, so new parameters must be selected for each one in order to achieve the best possible results.

8 Conclusion

In the last part of this work, we will shortly summarize our results and findings from chapter 6 and 7. To complete our elaboration, we will list up some open problems, which can be investigated in the future.

8.1 Summary

In this work, we investigated the influence of fine-tuning of pre-trained models on the performance of DP-SGD. We have given a brief overview of the contrastive learning method SimCLR and the principle of differential privacy. We discussed the issue with deep neural networks and presented Resnet as a solution. Furthermore, we presented a pipeline that allows us to train models and then evaluate them using DP-SGD. To do this, we used a two-step approach. Based on a Resnet50 architecture, we have trained a wide variety of models using different data sets. These then served as the basis for our DP fine-tuning and thus for the evaluation of the performance.
In our experiments, we then trained a linear layer based on these models using differential privacy. To do this, we iterated the images through the model and used the resulting embeddings, including labels, to train the linear layer supervised using DP-SGD. We found out that all models achieved similar results for the same tasks. Furthermore, the accuracy never or only marginally exceeded that of a model that only always chooses the largest of the available classes. After that, we presented the parameters that we used for DP-SGD. In doing so, we showed that we were able to achieve the desired security goals.
Summarized, it can be said that our models were not able to reliably fulfill the given tasks. In order to investigate what causes this could have, we used different approaches. We discussed whether the size of the training images has an impact on the accuracy of the model. We were able to determine that there is a marginal improvement, but we still cannot achieve sufficient separation of the classes. Likewise, it was discussed that other influences can also ensure that pre-training deteriorates. So the choice of transformations could pose a problem.
Furthermore, we examined the training curve of DP-SGD and showed that this can be a sign that the linear layer is trying to achieve separation in the dataset but is not able to do so. Reasons for this could be that the properties of our chosen evaluation data set are too difficult to be learned from our structure. However, it is also possible that the architecture

itself is not sufficient to be able to extract features well.
In order to investigate this further, we carried out an evaluation using a different data set. This also showed that reliable results could not be obtained. In addition, we examined how well our architecture performs after supervised training. We found that this does not increase the accuracy any further. We discussed that this can be an indication that our architecture is not sufficient.
To check what results we could achieve without the influence of DP, we also used a non-DP approach. For this, we evaluated our models in the fine-tuning step using Lars and showed that we can usually achieve better results, but these are also not good.
Finally, we discussed some challenges which we had defined at the beginning of the work and showed our solutions for it. In addition, we showed in each area how we adjusted the code to meet our needs.

8.2 Future Work

In this section, we want to show various possibilities to improve our approach and reach better results. For this, we fall back on our findings during the experiments.

8.2.1 Improvement Pre-Training

We have discussed that our results are mainly due to insufficient pre-training. Accordingly, this would have to be adapted and improved. On the one hand, it may be necessary for the training parameters to be better selected. On the other hand, other transformations may also be necessary in order to be able to reliably extract features from facial data.

8.2.2 Improvements DP-Optimizer

DP-SGD is just an optimizer that can ensure DP at the same time. There are now implementations that deliver better results. For example, DP-Adam. Based on the normal Adam optimizer, this one can also guarantee differential privacy. Therefore, one could investigate to what extent DP-Adam gives better results than DP-SGD.

8.2.3 Improvement Base Model

As we have seen, we used a Cifar10 model as the basis for some of our experiments. The motivation behind this was that we wanted to investigate whether prior knowledge of everyday things would give our models an advantage. Since Cifar10 only uses small images and is actually a fairly small data set, it may be a good idea to use generalized models for such an evaluation. An imagenet model could possibly deliver better results.

References

[ACG+16] Martín Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep Learning with Differential Privacy. *2016 ACM SIGSAC Conference on Computer and Communications Security (ACM CCS)*, pages pp. 308–318, 2016. arXiv preprint arXiv:1607.00133.

[ACP19] Galen Andrew, Steve Chien, and Nicolas Papernot. Tensorflow privacy. `https://github.com/tensorflow/privacy`, 2019. Accessed in November 2022.

[CCH14] Bor-Chun Chen, Chu-Song Chen, and Winston H. Hsu. Cross-age reference coding for age-invariant face recognition and retrieval. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014. `https://bcsiriuschen.github.io/CARC/`, Accessed in November 2022.

[CKNH20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *International Conference on Machine Learning (ICML) 2020*, 3, 2020. arXiv preprint arXiv:2002.05709.

[DR14] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3-4):pp. 211–407, 2014. DOI: 10.1561/0400000042.

[FSC20] William Falcon, Saurabh Saxena, and Ting Chen. Tf2 implementation of simclr. `https://github.com/google-research/simclr`, 2020. Accessed in November 2022.

[Goo15] Ian Goodfellow. Efficient Per-Example Gradient Computations. *Technical report, Google, Inc.*, 2015. arXiv preprint arXiv:1510.01799.

[HLW+22] Kunzhe Huang, Yiming Li, Baoyuan Wu, Zhan Qin, and Kui Ren. Backdoor Defense via Decoupling the Training Process. *International Conference on Learning Representations (ICLR) 2022*, 2022. arXiv:2202.03423.

[HRBLM07] Gary B. Huang, Manu Ramesh, Tamara Berg, and Erik Learned-Miller. Labeled faces in the wild: A database for studying face recognition in uncon-

strained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007. `http://vis-www.cs.umass.edu/lfw/`, Accessed in November 2022.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *ILSVRC & COCO 2015 competitions*, 1, 2015. arXiv preprint arXiv.1512.03385.

[LGW+21] Xiaoqiang Li, Chengyu Guo, Yifan Wu, Congcong Zhu, and Jide Li. Robust age estimation model using group-aware contrastive learning. *The Institution of Engineering and Technology*, 2021. DOI: 10.1049/ipr2.12552.

[LLWT15] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015. `https://mmlab.ie.cuhk.edu.hk/projects/CelebA.html`, Accessed in November 2022.

[LYW+22] Wenjun Li, Anli Yan, Di Wu, Taoyu Zhu, Teng Huang, Xuandi Luo, and Shaowei Wang. DPCL: Constrative Representation Learning with Differential Privacy. *Research Square*, 1, 2022. https://doi.org/10.21203/rs.3.rs-1516950/v1.

[RRL+18] Md.Atiqur Rahman, Tanzila Rahman, Robert Laganière, Noman Mohammed, and Yang Wang. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11:61–79, 2018. `https://www.tdp.cat/issues16/tdp.a289a17.pdf`.

[RTG18] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision*, 126(2-4):144–157, 2018. `https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/`, Accessed in November 2022.

[YGG17] Yang You, Igor Gitman, and Boris Ginsburg. Large Batch Training of Convolutional Networks. *Technical Report*, 2017. arXiv preprint arXiv:1708.03888.