universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Gradient Matching for Learning with Noisy Data“

verfasst von / submitted by

Lena Zellinger BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2022 / Vienna, 2022

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 645
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Data Science
Betreut von / Supervisor:	Univ.-Prof. Dr.-Ing. Benjamin Roth, B.Sc. M.Sc.

Acknowledgements

I would like to thank Univ.-Prof. Benjamin Roth and Anastasiia Sedova, MSc for their continued support and advice throughout the creation of this thesis. I want to stress that the method presented in this thesis was their idea; I merely built upon already existing work by Anastasiia.

Abstract

Deep neural networks can achieve excellent performance on various tasks when provided with enough data. However, large-scale datasets oftentimes contain annotation errors, which can negatively impact the model quality. To overcome this issue, so-called denoising methods, which facilitate training on error-prone data, can be applied.

In this thesis, a new denoising strategy based on the concept of "gradient matching" is presented. The goal of this method is to dynamically filter out or relabel mislabeled samples during the training process. Compared to some other denoising strategies, our algorithm is fairly simple and widely applicable, since it does not restrict the choice of the loss function used for training. Furthermore, the method is also suitable for multi-label datasets.

In order to evaluate the effectiveness of our denoising strategy, we conduct experiments on three noisy datasets. The results of the experiments show that the performance of our method strongly depends on the chosen loss functions and parameters. Moreover, the algorithm has a tendency to remove many correctly labeled samples from the batches. Nevertheless, an improvement in performance was observed in many cases.

Kurzfassung

Tiefe neuronale Netze können für verschiedenste Probleme exzellente Performance erreichen, wenn genügend Daten verfügbar sind. In großen Datensätzen finden sich jedoch häufig Annotationsfehler, welche die Modellqualität negativ beeinflussen können. Um diesen entgegenzuwirken, können sogenannte "Denoising Methoden", welche das Trainieren von Modellen auf fehlerbehafteten Daten vereinfachen, verwendet werden.

In dieser Arbeit wird eine neue Denoising Strategie basierend auf dem Prinzip von "Gradient Matching" vorgestellt. Ziel der Methode ist es, während des Trainingsprozesses Datenpunkte, welche einer falschen Klasse zugeordnet wurden, herauszufiltern oder zu korrigieren. Im Vergleich zu manch anderen Denoising Strategien ist unser Algorithmus relativ simpel und breit anwendbar, da er die Wahl der Verlustfunktion, mit der das Modell angepasst wird, nicht einschränkt. Darüber hinaus ist unsere Methode auch für Datensätze, welche mehrere Labels pro Instanz zulassen, geeignet.

Um die Effektivität unserer Denoising Strategie zu evaluieren, führen wir Experimente auf drei fehlerbehafteten Datensätzen durch. Die Ergebnisse der Experimente zeigen, dass die Performance unserer Methode stark von den gewählten Verlustfunktionen und Parametern abhängt. Darüber hinaus weist der Algorithmus eine Tendenz auf, viele korrekt annotierte Instanzen aus den Batches zu entfernen. Dennoch wurde in vielen Fällen eine Verbesserung der Performance observiert.

Contents

List of Tables

List of Figures

List of Algorithms

1. Introduction

The tremendous success of deep learning in recent years can partially be attributed to the increased availability of enormous labeled datasets, such as ImageNet [1, 2, 3, 4, 5]. As a result, it seems to be in our best interest to create massive training sets for diverse tasks and domains on which we can train powerful, parameter-rich models [2, 5]. However, annotating large-scale classification datasets is challenging. Depending on the size of the dataset, entrusting domain experts with the labeling process might not be feasible due to limited resources or time constraints. As a result, more and more datasets are annotated on the basis of pre-defined labeling functions, which efficiently assign labels to the training data in an automated fashion [2, 3, 6, 7]. This process is commonly referred to as *weak supervision*. The downside of this approach is that the resulting *weak labels* are usually error-prone [8, 6]. Labeling errors are however not exclusive to weakly supervised datasets but can also occur in expert-annotated data collections due to insufficient data quality or inherent subjectivity of the task [8]. As a result, *noisy labels* are prevalent in many widely-used benchmark datasets, such as MNIST [9] or ImageNet [10, 11]. Several authors have shown that training in the presence of mislabeled samples can negatively impact the model's generalization performance [12, 13, 10]. Therefore, so-called *denoising methods*, which can limit the negative effects of labeling errors on the training process, are commonly applied when working with data that presumably contains a large fraction of mislabeled samples [6]. The main goal of this thesis is to contribute a new, conceptually simple denoising strategy based on *gradient matching* [14, 15] which aims to dynamically filter out or correct likely mislabeled samples during training.

This thesis is divided into five main parts: In Chapter 2, a selection of previous work on the topic of learning with noisy labels is presented in order to put our approach into perspective. Moreover, the general concept of gradient matching is explained and some recent applications of the method are outlined. Chapter 3 provides the intuition behind the new, gradient-based denoising strategy explored in this thesis as well as its concrete formulation. In Chapter 4, the effectiveness of the proposed method is evaluated on two weakly supervised single-label datasets from the WRENCH benchmark [7], SMS [16, 17] and Youtube [18, 2]. Concretely, the performance of models trained with and without gradient-based denoising is contrasted. Furthermore, it is assessed how successful our approach is in identifying mislabeled samples by comparing the label changes induced by the denoising strategy with the ground-truth annotations provided by WRENCH [7]. A key benefit of our proposed framework, compared to some other approaches, is that any loss function can be used to update the model. Therefore, we also investigate the effects of losses based on smooth approximations of the F_1 score, after Bénédict et al. (2021) [19], on both the baseline performance and our denoising strategy. Chapter 5 repeats similar experiments on the CheXpert dataset, introduced by

Irvin et al. (2019) [20], which consists of chest radiographs annotated with respect to 12 pathologies. Unfortunately, this data collection does not contain expert annotations for the training set, so no direct comparison to the gold labels is possible. However, observing the performance achieved by our approach with respect to various multi-label evaluation metrics still gives valuable insights into the potential benefits and pitfalls of the method. Lastly, Chapter 6 summarizes the insights gained throughout the thesis and suggests some possible avenues for future work to further improve our proposed method.

2. Background and Related Work

The term *"noisy data"* generally refers to datasets with corrupted input features and/or incorrect labels [21]. This thesis solely focuses on label noise in classification tasks.

In the following, some previous studies on the characterization of label noise and its effect on the resulting model quality will be summarized. Then, a selection of already existing approaches for training well-performing models despite error-prone labels will be outlined.

2.1. Sources of Annotation Errors

There are many factors in the data collection process that can lead to incorrect labels, even when the data is annotated by human domain experts [8]. Frénay and Verleysen (2014) [8] determined the following main sources of label noise in the strongly supervised setting, where labels are usually assumed to be correct:

First of all, the input data, that is used to deduce the label, might be of low quality or lack relevant information [8]. As an example, Brodley and Friedl (1999) [22] mention missing medical test results which would be needed to confidently diagnose a patient. Secondly, depending on the task, experts might not reach a consensus on the appropriate annotation [8]. For example, in medical imaging, doctors might disagree on the diagnosis and, by extension, the suitable label(s) for the corresponding image [8, 23]. Other potentially subjective tasks include sentiment analysis and hate speech detection [24]. Lastly, the annotators might misinterpret imprecise instructions and consequently assign labels according to a wrong definition of the task [8].

In addition to the error sources identified by Frénay and Verleysen (2014) [8], Russakovsky et al. (2015) [4] observed that annotators had difficulties labeling the ImageNet classification task due to its high number of classes. For this benchmark, each training image was assigned to exactly one out of 1000 distinct labels [4]. Choosing the one correct label from such a large pool can be tricky, especially when classes are closely related or multiple objects are present in the image [10, 4].

Due to all of the factors mentioned above, labeling errors are prevalent in many widely-used, human-annotated datasets. Northcutt et al. (2021) [10] identified several incorrectly labeled samples in ImageNet and MNIST by using their "Confident Learning" framework, which is oftentimes referred to as *Cleanlab* due to the corresponding python package. Systematically finding annotation errors in massive datasets is difficult. Therefore, the mislabeled samples in the aforementioned data collections previously did not receive much attention, despite these datasets being popular benchmarks for new methods [10, 11, 4].

Nowadays, more and more datasets are annotated by automated processes, as opposed

to domain experts, in order to satisfy the demand for large-scale training sets [2, 3, 6]. This labeling strategy is commonly referred to as *weak supervision*. It allows for time-efficient annotation of the training data, however, a considerable portion of the resulting labels might be incorrect [2, 3, 6]. This setting was also mentioned by Frénay and Verleysen (2014) [8] as a frequent source of label noise.

The next section aims to give a brief overview of the concept of weak supervision, since automatically labeled data will be used for the experiments conducted in this thesis.

2.1.1. Weak Supervision

Manually annotating huge amounts of data can be very time-consuming and expensive, especially when domain-specific knowledge is required to assign suitable labels to the samples, like in the medical domain [20, 25, 26, 27, 6]. However, sufficiently large datasets are essential for training modern supervised machine learning models, which oftentimes have millions of learnable parameters [28, 5]. One possible solution to this problem is annotating datasets with *weak supervision* [2, 3, 6]: In the weakly supervised setting, the training data is labeled by automated annotation mechanisms instead of human domain experts. Commonly, a set of labeling functions, which map the input to a certain label, is defined [2, 3, 6]. These functions can be designed on the basis of various sources such as [2, 3, 6]:

- information from external databases (*"distant supervision"*)
- *heuristics* which aim to represent expert knowledge
- crowd-sourced annotations (e.g. from Amazon Mechanical Turk)

The resulting annotation rules can be simple keyword rules or more intricate rules that represent patterns with regular expressions [2, 3, 6]. For the task of identifying spam in the comment section of music videos published on YouTube, Ratner et al. (2020) [2] defined the following rules[1], among others, to label the training data [29].

KEYWORD_SONG → ham (0)
KEYWORD_SUBSCRIBE → spam (1)

The above rules express that if a comment contains the keyword "song", it should be mapped to the negative label, while if the text contains "subscribe", that is an indication for a spam comment [29, 2]. Depending on which rules are chosen for the task, the quality of the resulting labels can vary greatly [2]. Curating a suitable set of labeling functions is challenging [2, 3, 6]: If the rules have too little coverage, meaning that only few instances fulfil the condition that the rule expresses, a lot of samples might remain unlabeled, raising the question how to handle them. On the other hand, a rule with very

[1]The full set of rules can be found at `https://www.snorkel.org/use-cases/01-spam-tutorial`, last accessed 2022-05-20

high coverage, that matches almost all samples, might conflict with other rules and fail to effectively distinguish classes [2, 3, 6]. Moreover, a tie can arise when multiple rules match for a single example [2, 3, 6]. Consider the two annotation rules mentioned before [29]: If a comment contains both "song" and "subscribe", which label should be chosen? Oftentimes, one of the tied labels is randomly assigned to the sample in these cases, which can easily lead to annotation errors [6]. Even when using a well-designed set of labeling functions, at least some of the final labels will likely be inaccurate, especially when it comes to huge datasets [10, 2, 3, 6].

All in all, weak supervision allows to efficiently annotate large amounts of data, however, the resulting labels might not match the input [2, 3, 6].

The prevalence of labeling errors in both human-annotated and automatically labeled datasets raises the question whether training a model on noisy labels deteriorates its generalization performance and if so to which extent. However, the effect of corrupted labels on the final model performance has largely been studied on datasets with synthetic label noise, as opposed to naturally error-prone datasets, like those obtained from weak supervision [12]. The following section therefore gives a short overview of different types of label noise that are commonly simulated in the literature.

2.2. Types of Label Noise

Previous studies on learning with noisy labels primarily distinguish between *uniform*, *class-dependent* and *feature-dependent* noise (cf. Algan and Ulusoy 2020, Frénay and Verleysen 2014) [12, 8].

In the *uniform* (or *symmetric*) noise model, the probability of mistaking a sample with true class y^* for being of class $\widetilde{y} \neq y^*$ is the same for all combinations of y^* and $\widetilde{y}$. More precisely, assume that each sample is mislabeled with probability ρ and that there are K possible labels [12, 8]. Then

$$P(\widetilde{y} = j | y^* = k) = \begin{cases} 1 - \rho, \, j = k \\ \frac{\rho}{K-1}, \, j \neq k \end{cases}$$

under the assumption of uniform noise (cf. Algan and Ulusoy 2020, Frénay and Verleysen 2014) [12, 8]. This type of noise is very easy to generate, however, it does not accurately reflect real-world label noise [12, 8, 13].

The *class-dependent* (or *asymmetric*) noise model takes into account that some pairs of classes are harder to distinguish than others [12, 8]. The resulting probabilities of label y^* being confused with $\widetilde{y}$ are usually represented in a *noise transition matrix* of the form

$$\begin{pmatrix} P(\widetilde{y} = 1 | y^* = 1) & \dots & P(\widetilde{y} = K | y^* = 1) \\ \vdots & \ddots & \vdots \\ P(\widetilde{y} = 1 | y^* = K) & \dots & P(\widetilde{y} = K | y^* = K) \end{pmatrix}$$

where K is again the total number of classes (cf. Algan and Ulusoy 2020, Frénay and Verleysen 2014) [12, 8]. Note that in this setting, the probability of mislabeling a sample

as a certain class only depends on its true label, but not its features, which is not always realistic [12, 8].

Therefore, this assumption is removed in the *feature-dependent* setting, where the probability of assigning an instance to an incorrect class hinges on its features [12, 30, 8]. Intuitively, it makes sense that the probability of mistaking a sample belonging to class y^* for class $\widetilde{y}$ would depend on the actual realization of the sample [12]. Algan and Ulusoy (2020) [12] give the following example:

> "some sport cars are more similar to classic cars than others" [12, Introduction]

As a result, these particular sport cars are more likely to be mislabeled as a classic car by the annotation process than other members of this class [12]. Feature-dependent noise most closely models real-world label noise, however, compared to the other two settings, its effect on the model performance has not been studied as thoroughly, since it is harder to simulate [12, 8].

Several authors have conducted experiments that assess the influence of noisy labels on the quality of the resulting model. The next section summarizes some interesting findings.

2.3. Effect of Noisy Labels on the Generalization Performance

Past studies on the effect of error-prone labels on the final model performance have provided divisive results, depending on the examined noise type and general setup of the experiment.

According to Rolnick et al. (2018) [27], deep neural networks are naturally robust to label noise. They showed this by augmenting well-known datasets, such as MNIST, CIFAR-10 [31] and ImageNet, by large numbers of noisy examples and monitoring the resulting drop in performance for various deep neural network architectures. Both uniform and class-dependent noise was examined in their experiments. They found that very deep-layerd neural networks generally learn the respective task reasonably well, despite noisy samples outnumbering the clean training examples by a large margin. It should however be stressed that in their experiments, all original, presumably clean training samples were preserved and mislabeled samples were simply added on top of the already existing dataset, which is far removed from the weak supervision setting. Moreover, Rolnick et al. (2018) [27] stated that a sufficient amount of clean training data is essential to obtain good performance despite the presence of mislabeled samples. In fact, they found the removal of clean instances to be more harmful to the generalization performance than the introduction of additional mislabeled samples [27].

Algan and Ulusoy (2020) [12] took a different approach for assessing the effects of label noise: Instead of adding noisy samples to the pre-existing dataset, they corrupted 5% to 75% of the original labels and observed the resulting decrease in performance. Moreover, they also introduced a simulation scheme for feature-dependent label changes, which are oftentimes neglected in other studies. In essence, the corruption strategy proposed by Algan and Ulusoy (2020) [12] identifies samples which have similar representations to

instances of a different class and changes their label accordingly. Algan and Ulusoy (2020) [12] compared their noise generation algorithm to a method introduced by Inouye et al. (2017) [13], which simulates *locally concentrated* noise. To create locally concentrated noise, Inouye et al. (2017) [13] proposed to sample some instances from the training set and determine their neighborhoods according to a k-nearest neighbors algorithm. The labels of all data points in a given neighborhood are then systematically perturbed so that, in the end, each cluster has a unified noisy label. When comparing the effect of different noise types on MNIST-Fashion [32], Algan and Ulusoy (2020) [12] found feature-dependent noise to be more detrimental to the test scores than uniform or class-dependent noise. In their experiments on CIFAR-100 [31], corruption with locally concentrated noise resulted in the worst generalization performance, followed by feature-dependent noise generated according to their simulation strategy [12]. The results obtained by Algan and Ulusoy (2020) [12] on MNIST-Fashion and CIFAR-100 suggest that real-world label noise might be more harmful than simulated uniform or class-dependent corruptions.

Northcutt et al. (2021) [10] studied the effects of erroneous annotations in widely-used benchmark datasets. Using their Cleanlab framework, they identified and removed likely mislabeled samples in the ImageNet data collection and compared the performance of a model trained on the original dataset to a classifier fit to the reduced version. They found that training on the cleaned dataset improved the validation performance. This result by Northcutt et al. (2021) [10] demonstrates that real-world label noise can indeed limit the model quality.

All of the results stated above, except for the findings of Rolnick et al. (2018) [27], suggest that noisy training data can have a significant negative impact on the generalization performance. Therefore, it can be highly beneficial to apply *denoising methods* when working with imperfect labels [2, 6].

Following Sedova et al. (2021) [6], a denoising method within the context of this thesis is any method aiming to boost performance when training with error-prone labels [6]. The main contribution of this thesis is the exploration of a novel denoising strategy based on gradient matching. In order to put this new method into perspective, the next section shortly outlines some previously proposed approaches for learning with noisy labels.

2.4. Some Existing Denoising Frameworks

There are multiple general strategies for dealing with label noise [10, 6]. Northcutt et al. (2021) [10] distinguish between two main approaches: The *"model-centric"* approach focuses on modifying the model architecture or the loss function to facilitate learning with noisy data. The *"data-centric"* approach, on the other hand, tries to identify likely mislabeled samples and subsequently removes or corrects them before training the final model on the dataset [10]. In order to clarify these two concepts, representatives of each approach will briefly be introduced. Please note that what follows is a very narrow selection of possible methods to handle label noise, aiming to highlight some of the general directions that have been studied so far.

2.4.1. Model-centric Approach

Several authors have explored dealing with noisy labels via novel or surrogate loss functions. Formally, Ghosh et al. (2017) [33] define a loss as *noise-tolerant* if the classifier that minimizes the risk with respect to this loss is the same on clean and on corrupted data. They showed theoretically that the mean absolute error (MAE) loss fulfills this condition under uniform noise and, if certain assumptions are met, under class-dependent noise. Empirically, they observed that models trained with the MAE had better generalization performance than their counterparts fit with the common cross-entropy loss in various noise settings. However, Ghosh et al. (2017) [33] noticed that training a network solely with the MAE loss leads to much slower convergence compared to when the standard cross-entropy loss is used [33].

As a result, instead of using a noise-tolerant loss by itself, some authors alter already existing loss functions by adding a noise-tolerant term. For example, Wang et al. (2019) [34] modified the cross-entropy loss, because they observed *class bias* when training with this criterion: Some classes are learned much faster than others, leading to over- and underfitting respectively. They found that noisy labels amplify this effect and therefore proposed the *symmetric cross-entropy* loss, which adds a noise-robust term, the *reverse cross-entropy*, to the original cross-entropy loss. Combining the two terms is crucial, since models trained with the stand-alone reverse cross-entropy converged fairly slowly in the experiments of Wang et al. (2019) [34], which mirrors the results that Ghosh et al. (2017) [33] obtained with the MAE loss.

Other approaches define losses that encourage the model to reject likely mislabeled samples during training [1, 35, 36]. For example, the method proposed by by Ziyin et al. (2020) [36] allows the model to ignore a data point during training if its *"uncertainty score"* is too high. This score is predicted by adding an extra output node to network, representing the option of rejecting a sample, and training this architecture with a special loss function, called the *gambler's loss* [36]. In experiments conducted on MNIST, Ziyin et al. (2020) [36] showed that the trained model learned to assign higher rejection scores to mislabeled samples than to correctly labeled samples. While the concrete method proposed by Ziyin et al. (2020) [36] is vastly different from the one explored in this thesis, the general idea of dynamically ignoring unreliable samples during training has parallels to our approach.

Instead of adapting the loss, Sukhbaatar et al. (2014) [37] proposed adding a so-called *"noise layer"* after the classification head of convolutional neural networks. The weights of this additional layer represent an estimate of the noise transition matrix, under the assumption of asymmetric label noise, that is dynamically learned throughout the training process [37]. The inclusion of this layer allows to adjust the predicted probabilities according to the estimated noise distribution before they are passed to the loss [38, 39, 37].

While the previously described approaches consider all samples in the training set to be potentially noisy, Hendrycks et al. (2018) [40] proposed leveraging a small set of strongly supervised samples to estimate the noise transition matrix [40, 39]. Akin to the method proposed by Sukhbaatar et al. (2014) [37], the estimated noise transition matrix is then multiplied with the model outputs of samples with unverified labels before the

cross-entropy loss is applied [40, 39, 37].

The approaches outlined above make fairly simple alterations to the loss function or the model architecture. However, there are also more complex frameworks for learning with noisy labels that are for example based on ensemble methods or meta-learning [1, 41, 42, 43].

As is evident from the examples above, a lot of work has gone into developing methods that handle the presence of noisy labels dynamically during training. In contrast, Northcutt et al. (2021) [10] propose the Cleanlab framework, which is a strictly data-centric approach that identifies and removes likely mislabeled samples *before* the final model is trained. Unlike the previously described denoising strategies, Cleanlab [10] therefore separates the denoising process and the training of the final model. As a result, once the dataset was cleaned by the procedure, any (suitable) model architecture and loss function can be used to train systems on the data [10]. The following section briefly sketches the Cleanlab procedure[2].

2.4.2. Data-centric Approach

The Cleanlab framework, developed by Northcutt et al. (2021) [10], follows a data-centric approach. The goal is to identify likely mislabeled samples in a given dataset and provide a "clean" version of said dataset, which can then be used for training. Cleanlab is based on the assumption of class-dependent noise as it was described in Section 2.2: The probability that a sample is wrongly assigned label $\widetilde{y}$ by the annotation process only depends on its true (unknown) label y^* and not on the corresponding input data [12, 8, 10, 11]. Intuitively, Northcutt et al. (2021) [10] justify this assumption with the following example:

> "a *leopard* is more likely to be mislabeled *jaguar* than *bathtub*" [10, p. 1374]

Under this assumption, the joint distribution between the noisy label $\widetilde{y}$ and the true label y^* can be estimated using only the noisy labels and out-of-sample model outputs for each data point [10]. Northcutt et al. (2021) [10] recommend using cross-validation for obtaining these outputs. On the basis of the estimated joint distribution, the total number of labeling errors in the dataset can be gauged and diverse ranking and pruning methods can be applied to remove likely mislabeled samples from the dataset. The final model is then trained on the clean version of the dataset. Using this strategy, Cleanlab has been successful in finding labeling errors in both the training and the test sets of popular benchmark data collections such as MNIST and ImageNet [10, 11].

A key benefit of the Cleanlab framework is that the predicted probabilities used for estimating the joint distribution can be retrieved from any model architecture [10]. However, since the resulting distribution is the method's main basis for identifying mislabeled samples, the chosen model should already provide reasonably good outputs [10]. As a result, the overall effectiveness of the method is to an extent dependent on the quality of this particular model [10].

[2]The entire following subsection is based on Northcutt et al. (2021) [10].

As can be gathered from the last section, there already exist well-performing denoising frameworks to choose from. However, many representatives of the model-centric approach dictate the loss that is used for training, while the data-centric approach by Northcutt et al. (2021) [10] requires training several different models to obtain out-of-sample predictions for each data point. This thesis contributes the exploration of a novel, comparatively simple denoising strategy based on *gradient matching* that dynamically cleans the data during training and can be used with a variety of loss functions.

Gradient matching in itself is not a new concept but has been studied before in different contexts, such as domain generalization and dataset condensation [14, 15]. The next section therefore gives a brief overview of the general idea of gradient matching and some previous applications of the concept.

2.5. Main Idea of Gradient Matching

Gradient matching (GM) in itself is a fairly simple concept. In principle, gradient matching compares model gradients that were computed on different parts of the data or on distinct datasets [14, 15]. Usually, the inner product or the cosine similarity is used to assess the alignment of the gradients [14, 15].

The general idea of gradient matching recently gained some traction with several papers utilizing gradient matching being published in 2021 [14, 15]. The following section will briefly summarize lately proposed applications of gradient matching, demonstrating the versatility of the method.

2.5.1. Previous Applications of Gradient Matching

Domain Generalization

Shi et al. (2021) [14] successfully made use of gradient matching for *domain generalization* on multiple benchmark datasets. The objective of domain generalization is to create models that achieve good performances on test sets that significantly differ from the training sets in terms of the nature and/or origin of their instances [14]. Examples of such datasets can be found on the WILDS benchmark [44]. The CAMELYON17-WILDS dataset [45, 44] can be used to to assess how well a breast cancer classifier trained on images from selected hospitals performs on images from different hospitals. Good generalization is especially important for this application, since in practice, the classifier would likely be applied in several hospitals and not just in the ones that provided the training data [44, 14]. Shi et al. (2021) [14] tackled the mismatch of training and test domains by defining the *"inter-domain gradient matching (IDGM) objective"*, which promotes the alignment of model gradients obtained from different training domains by maximizing their inner product. Their core idea was the following: If the gradients obtained from two parts of the data with different domains are similar, then moving in the direction of either gradient should increase the performance on both domains. As a result, the model should learn what Shi et al. (2021) [14] call *"invariant features"*, namely features that are not affected by the domain change [44, 14]. By optimizing the

IDGM objective with their computationally efficient *Fish* algorithm, Shi et al. (2021) [14] managed to achieve superior results compared to the (previously) best-performing methods on 4 out of 6 selected datasets from the WILDS benchmark, showcasing that gradient matching is a powerful tool for domain generalization [44, 14].

Dataset Condensation

Another newly explored application of gradient matching is *dataset condensation*, where the goal is to reduce a large dataset to a comparatively small number of representative samples [15]. Zhao et al. (2021) [15] approached the problem by generating artificial data instances instead of selecting samples from the original data. They used the following technique [15]: After initializing the examples for the condensed dataset from real training instances or noise, the samples are updated by comparing the gradients obtained from training on the fabricated instances to the ones computed on the original data. In particular, the synthetic samples are altered to increase the cosine similarity of these gradients [15]. A model trained on the resulting small set of synthetic examples should then achieve comparable performance and learn similar parameters to a system of the same architecture trained on the original large dataset [15]. Zhao et al. (2021) [15] evaluate their method on four image datasets: MNIST, SVHN [46], MNIST-Fashion and CIFAR-10. On all four data collections, the method managed to outperform popular coreset methods, such as K-Center. Moreover, on MNIST, the resulting performance was especially close to the model trained on the entire dataset [15].

The two previous applications of gradient matching outlined above suggest that gradient comparisons can be a simple, yet powerful tool for tackling complex problems. The recent success of gradient matching on diverse tasks raises the question whether it is possible to formulate a denoising method based on gradient matching to boost performance on noisy datasets. In the next chapter, this thesis' attempt at formalizing such a method will be outlined.

3. Denoising with Gradient Matching

This section gives an overview of the new denoising method proposed in this thesis. After the basic intuition behind the approach is clarified, the concrete formulation of the algorithm is provided. Moreover, the method will be contrasted with some of the approaches outlined in Section 2.4 to emphasize why our method could be a valuable contribution to the field.

3.1. Intuition

The goal of the developed denoising algorithm is to dynamically filter out or correct likely mislabeled instances during training. In order to achieve that, in each batch, the individual model gradients of the samples are computed and compared with an aggregated gradient from another batch of the data, the so-called *"comparison batch"* with its *"comparison gradient"*. Assuming that the overall noise rate in the dataset is not too high and the batch size is reasonably large, this aggregated gradient could be seen as an expected weight change on mostly clean data. If the individual gradient of a sample and the comparison gradient point into opposing directions, this could be an indication that the sample is mislabeled. There are two options on how to proceed with this sample:

1. remove the sample from the batch so that it does not influence the weight update
2. assign a different label to the sample so that its gradient aligns with the comparison gradient

Note that, using this method, the mislabeled samples do not have to be decided upon before training the model, but which samples are removed or corrected can dynamically change during the training process. This may seem counterintuitive from the standpoint that each label has only one ground-truth, correct or incorrect. However, this approach has the advantage that only a single model has to be trained, unlike in the Cleanlab framework [10]. Moreover, this strategy has an opportunity to "correct its own mistakes": If one commits to a final, cleaned dataset before starting the training process, missed labeling errors will affect the entire training procedure and mistakenly removed samples cannot be reincorporated at any point. Our gradient-based method, on the other hand, allows to remove samples on the fly, on the basis of the current state of the model, and can reconsider keeping or ignoring a particular instance in each epoch.

The following section gives some more details on the steps of the algorithm and its parameters.

3.2. Method Details

3.2.1. Gradient Comparison

The similarity of the model gradients obtained from the update batch and the comparison batch will be measured by the cosine similarity, following an implementation[1] by Zhao et al. (2021) [15]. The concrete formula is given by

$$sim(g_{(X_t,y_t)}, \widetilde{g}) = \frac{g_{(X_t,y_t)} \cdot \widetilde{g}}{||g_{(X_t,y_t)}||_2 \, ||\widetilde{g}||_2 + 0.000001} \tag{3.1}$$

where $g_{(X_t,y_t)}$ denotes the concatenated model gradients for sample t with feature set X_t and label y_t, while $\widetilde{g}$ denotes the flattened gradient computed on the comparison batch (cf. Zhao et al. 2021) [15]. The small constant in the denominator prevents divisions by 0 and was also added in the implementation by Zhao et al. (2021) [15]. In the following, $sim(g_{(X_t,y_t)}, \widetilde{g})$ will generally be referred to as the *similarity score* or the *gradient similarity* of sample t.

3.2.2. Parameter Choices

Denoising with gradient matching is a fairly simple and versatile concept. Several design choices can be made to adapt the algorithm to the specific dataset it should be applied to. This section briefly summarizes the main input parameters of the proposed algorithm and their effect on the procedure.

Alternative Label

One of the core decisions is the specification of an alternative label which can be assigned to a sample if this results in a better gradient similarity, computed according to Equation 3.1, than using the sample's original annotation. If such a label is not included, an instance will be ignored if its similarity score is smaller than or equal to a threshold τ. If an alternative label is specified, the algorithm will compute this similarity with both labels and

1. keep the original label if this achieves the highest similarity score and the score is greater than τ
2. change the label of the sample to the alternative label if this gives the best similarity score and the score is greater than τ
3. ignore the sample if neither similarity score is greater than τ

Which label could make sense as the alternative label depends on the specific dataset. One possible choice could be a negative label, like "no relation" for the task of relation extraction or "no finding" for medical imaging [20, 47].

[1] The code by Zhao et al. (2021) [15] is available at `https://github.com/VICO-UoE/DatasetCondensation/blob/master/utils.py`, last accessed 2022-06-29

Removal Threshold

The removal threshold τ dictates how small the gradient similarity must at least be in order for a sample to be ignored for the batch update. Since samples with opposing gradients to the comparison gradient should be removed, it makes sense to choose $\tau \in (-1, 0]$, given Equation 3.1. Low values of τ, close to -1, only remove samples with gradients that are strongly dissimilar to the comparison gradient. Higher threshold values generally tend to ignore more samples during training.

Comparison Batch Sampling

Since the same comparison batch is used to evaluate all instances in a given update batch, it is important to sample it in a way that does not systematically disadvantage examples of a specific class. For datasets with a fairly even class distribution, randomly drawing a comparison batch from the training data might be sufficient to get a well-balanced comparison batch and consequently an aggregated gradient that takes all classes into account. However, for imbalanced datasets, class-weighted sampling might be beneficial to ensure a large enough fraction of minority class instances in the comparison batch.

Update and Comparison Loss

Our proposed denoising strategy does not put any restrictions on the used loss functions. In principle, the loss that is used for computing the compared gradients can differ from the update criterion. The losses will be referred to as the *comparison loss* and the *update loss* respectively. Whether choosing a different loss function for the gradient comparison and the update can be beneficial is examined in our experiments.

Now that the main concept of the algorithm was introduced, the next section provides the concrete formulation of the procedure.

3.3. Algorithm Formulation

The denoising method explored in this thesis can be applied to both single-label and multi-label problems. Pseudocode for each scenario can be found in this section[2]. For the rest of the thesis, the proposed method will be called the *GM algorithm* for simplicity.

3.3.1. Notation

Table 3.1 summarizes the notation used for describing the GM algorithm throughout the thesis.

[2]The formulations of the two algorithms were inspired by Yu et al. (2020) [48]

Expression	Definition
$f(\cdot;\theta)$	model with parameter set θ
$\mathcal{D}_T$	training set
E	total number of epochs
M	batch size
K	number of classes
$\mathcal{B}$	original update batch
$\widetilde{\mathcal{B}}$	comparison batch
$\mathcal{B}^*$	update batch after the changes induced by the GM algorithm
$\mathcal{L}$	update loss
$\widetilde{\mathcal{L}}$	comparison loss
a	alternative label that can be assigned by the GM algorithm
i	label for samples that should be ignored in the loss computation
$g_{(X_t,y_t)}$	flattened model gradient for sample t with feature set X_t and label y_t
$\widetilde{g}$	flattened comparison gradient

Table 3.1.: Basic Notation for the GM Algorithm

3.3.2. Single-Label Setting

In the single-label case, each sample is associated with exactly one label [49]. In this setting, the GM algorithm can either ignore a likely mislabeled sample entirely by removing it from the batch or correct its annotation to an alternative label a.

3.3.3. Multi-Label Setting

In the multi-label setting, each instance can belong to multiple classes [49]. Many real-world applications allow for multi-label annotations. In tasks like topic, genre, or scene classification, multiple labels can simultaneously be positive for a single instance [49]. For example, a movie can incorporate elements of both the "Action" and the "Comedy" genre [50].

In the formulation of the multi-label version of the GM algorithm (cf. Algorithm 2), y_t is a binary vector of length K. If the k-th entry of this vector, $y_{t,k}$, is 0, the sample was not associated with class k by the annotation process, while $y_{t,k} = 1$ indicates that the sample presumably belongs to this class. In the multi-label version of the GM algorithm, the gradient comparison happens between gradients associated with each individual class. Therefore, individual elements of the K-dimensional label vector can be ignored. The label-specific gradients will be denoted by $\left(g_{(X_t,y_t)}\right)_k$ and $(\widetilde{g})_k$, $k \in \{1, ..., K\}$. For the experiments in this thesis, the multi-label setting does not feature an alternative label nor weighted sampling for the comparison batch.

Input : training set $\mathcal{D}_T$,
initial model $f(\cdot;\theta)$,
alternative label a,
removal threshold τ
Output : trained model $f(\cdot;\theta^*)$
for *e = 1,...,E* **do**
  foreach *batch* $\mathcal{B}$ **do**
    Sample a comparison batch $\widetilde{\mathcal{B}}$ of size M from $\mathcal{D}_T$;
    Compute $\widetilde{g}$ on $\widetilde{\mathcal{B}}$ wrt. $\widetilde{\mathcal{L}}$;
    Set up corrected batch $\mathcal{B}^* \leftarrow \mathcal{B}$;
    forall *(X_t, y_t)* $\in \mathcal{B}$ **do**
      Compute $g_{(X_t,y_t)}$ wrt. $\widetilde{\mathcal{L}}$;
      Compute $\mathrm{sim}(g_{(X_t,y_t)}, \widetilde{g})$ according to Equation 3.1;
      if *an alternative label a is specified* **then**
        Compute $g_{(X_t,a)}$ wrt. $\widetilde{\mathcal{L}}$;
        Compute $\mathrm{sim}(g_{(X_t,a)}, \widetilde{g})$ according to Equation 3.1;
        if $sim(g_{(X_t,y_t)}, \widetilde{g}) \leq \tau$ *and* $sim(g_{(X_t,a)}, \widetilde{g}) \leq \tau$ **then**
          $\mathcal{B}^* \leftarrow \mathcal{B}^* \setminus \{(X_t, y_t)\}$
        end
        if $sim(g_{(X_t,a)}, \widetilde{g}) > \tau$ *and* $sim(g_{(X_t,a)}, \widetilde{g}) \geq sim(g_{(X_t,y_t)}, \widetilde{g})$ **then**
          $\mathcal{B}^* \leftarrow (\mathcal{B}^* \setminus \{(X_t, y_t)\}) \cup \{(X_t, a)\}$
        end
      else
        if $sim(g_{(X_t,y_t)}, \widetilde{g}) \leq \tau$ **then**
          $\mathcal{B}^* \leftarrow \mathcal{B}^* \setminus \{(X_t, y_t)\}$
        end
      end
    end
    Update θ wrt. $\mathcal{L}$ and $\mathcal{B}^*$
  end
end

Algorithm 1: GM Algorithm for Single-Label Datasets

Input : training set $\mathcal{D}_T$,
initial model $f(\cdot;\theta)$,
removal threshold τ
Output : trained model $f(\cdot;\theta^*)$
for *e = 1,...,E* **do**
  foreach *batch* $\mathcal{B}$ **do**
    Sample a comparison batch $\widetilde{\mathcal{B}}$ of size M from $\mathcal{D}_T$;
    Compute $\widetilde{g}$ on $\widetilde{\mathcal{B}}$ wrt. $\widetilde{\mathcal{L}}$;
    Set up corrected batch $\mathcal{B}^* \leftarrow \mathcal{B}$;
    forall $(X_t,\ y_t) \in \mathcal{B}$ **do**
      Compute $g_{(X_t,y_t)}$ wrt. $\widetilde{\mathcal{L}}$;
      Set up corrected label vector $y_t^* \leftarrow y_t$;
      for *k=1,...K* **do**
        Compute $\text{sim}(\left(g_{(X_t,y_t)}\right)_k,\ (\widetilde{g})_k)$ according to Equation 3.1;
        if $sim(\left(g_{(X_t,y_t)}\right)_k,\ (\widetilde{g})_k) \leq \tau$ **then**
          $y_{t,k}^* \leftarrow i$
        end
      end
      $\mathcal{B}^* \leftarrow (\mathcal{B}^* \setminus \{(X_t, y_t)\}) \cup \{(X_t, y_t^*)\}$
    end
    Update θ wrt. $\mathcal{L}$ and $\mathcal{B}^*$;
  end
end

Algorithm 2: GM Algorithm for Multi-Label Datasets

3.3.4. Potential Benefits of Denoising with Gradient Matching

The plethora of available denoising approaches might raise the question why the proposed method is potentially a valuable addition. The following paragraph therefore aims to outline some desirable characteristics of the GM algorithm.

First of all, as can be gathered from Algorithm 1 and Algorithm 2, the overall procedure is fairly simple. The entire denoising process happens *during* the training of a single model. Furthermore, unlike common model-centric approaches [33, 34, 36], our method does not put any restrictions on the loss that is used for training. Moreover, our approach is not based on the assumption of a particular noise type and does not require any gold annotations. Another key benefit of the method becomes apparent in the context of weak supervision: Several existing denoising strategies geared towards weakly supervised data, such as the *Snorkel* framework by Ratner et al. (2020) [2], require access to the rule matches for each sample [29]. Our approach does not use any information about the annotation process; only the resulting weak labels are required. Furthermore, the method's applicability to the multi-label setting is crucial for many tasks [49]. If the GM algorithm achieves good performance, by making reasonable label changes, it could provide a conceptually simple and widely applicable option for learning with noisy labels.

In order to assess the effectiveness of the method, it will be evaluated on two single-label datasets and one multi-label dataset, all of which are weakly supervised. The following two chapters describe the setup and the results of the respective experiments.

4. Experiments on Single-Label Datasets

This section evaluates the performance of our proposed denoising method, as described in Algorithm 1, on two weakly supervised single-label datasets retrieved from the WRENCH benchmark [7]. The performance of the GM algorithm will be compared against a baseline trained on the unaltered noisy labels. Moreover, since the WRENCH benchmark provides gold labels for most datasets [7], it will be assessed how successful gradient matching is in removing and/or correcting mislabeled samples. Furthermore, the behaviour of the GM algorithm will be inspected with respect to several design choices, namely the encoding of the data, the update loss, the comparison loss, the sampling strategy for the comparison batch, and the the removal threshold τ.

4.1. Data Description

Table 4.1 provides an overview over some characteristics of the two selected single-label datasets from the WRENCH benchmark[1] [7], SMS [16, 17] and Youtube [18, 2].

Data	Labels	Train	Dev.	Test	Labeling Functions
SMS	2	4571	500	500	73
Youtube	2	1586	120	250	10

Table 4.1.: Overview of Single-Label Datasets[2]

The following two sections briefly introduce the datasets.

4.1.1. SMS

The SMS dataset [16] is a collection of SMS messages gathered from various sources, including Grumbletext and already existing corpora. The task for this dataset is to predict whether a given SMS message is spam or not. The non-spam messages are commonly referred to as "ham" [16, 7].

Awasthi et al. (2020) [17] created 73 labeling functions for this dataset on the basis of a small set of strongly supervised examples. For each message, WRENCH provides the corresponding rule matches, which can be used to assign weak labels to the instances. Each sample is therefore annotated with a vector of 73 labels in the raw data files[3]

[1] The raw data files are available at `https://github.com/JieyuZ2/wrench`, last accessed 2022-05-23

[2] Adapted from Zhang et al. 2021 [7], p. 4

[3] A detailed description of the WRENCH dataset structure can be found at `https://github.com/JieyuZ2/wrench/wiki/Dataset:-Format-and-Usage`, last accessed 2022-07-05

[17, 7, 51]. If a rule matched the sample, the corresponding entry in this label vector is either 0 or 1, depending on the class this rule represents [7, 51]. Otherwise, -1 was assigned to this position in the label vector, indicating that the respective rule did not match the sample [7, 51]. Following Zhang et al. (2021) [7], the evaluation metric for this dataset is chosen to be the F_1 score of the positive class, because the dataset is imbalanced, as will be discussed in a later section.

4.1.2. Youtube

The second dataset, Youtube [16], is also a spam detection dataset. However, instead of SMS messages, it contains texts collected from the Youtube comment section of music videos by five different artists [18, 7]. In this context, a spam comment is a text unrelated to the video [18]. Oftentimes, such comments aim to promote the commenter's own YouTube channel or links to external websites [18, 29, 2]. Ratner et al. (2020) [2] manually curated 10 labeling functions[4] [29] for this dataset. The corresponding rule matches are again provided by WRENCH [7]. For this dataset, accuracy will be used as the evaluation metric, as there is no strong class imbalance.

Since the SMS and the Youtube dataset share the same task, namely spam classification, the experimental setup and basic data preparation was the same for both datasets.

4.2. Data Preprocessing

For the training samples, WRENCH only provides the rule matches, which the user has to convert to noisy labels themselves [7]. Therefore, both the raw texts and the annotations of the two datasets require preprocessing.

For the experiments in this thesis, *majority voting* was chosen to annotate each instance in the training set with a weak label. When using majority voting, the class which corresponds to the most rule matches for a particular text is assigned [6]. The resulting label distribution can be found in Table 4.2.

Data	Ham (0)	Spam (1)	Not Matched	Tied
SMS	1592	252	2719	8
Youtube	627	531	195	233

Table 4.2.: Label Distribution of the Training Set after Majority Voting

For the SMS dataset, many samples remain unlabeled, because none of the 73 rules matched them (cf. Awasthi et al. 2020) [17, 7]. We decided to remove such samples from both datasets for the experiments. This approach is likely suboptimal if the aim is to achieve a competitive performance compared to other published results on these datasets.

[4]The concrete rules can be found at https://www.snorkel.org/use-cases/01-spam-tutorial, last accessed 2022-05-20

However, since the primary goal of the following experiments is merely to compare the proposed denoising method to a simple baseline trained on the same data, this approach should be sufficient. Following Sedova et al. (2021) [6], ties will be handled by randomly assigning one of the tied labels. Note that, disregarding the unmatched samples, the SMS dataset is highly imbalanced, while the Youtube dataset has a comparatively high number of ties [7].

Since both datasets are text-based, there is a variety of options for encoding the input. Most prominently, one can choose between using sparse and dense representations of the texts [52, 53]. *Sparse encodings* are generally based on bag-of-word features [52]: For each text, the occurrence of all terms in the vocabulary is counted. These counts are then used to form a vector, which is of the same size as the vocabulary, to represent the corresponding text [52]. Since the vocabulary usually contains many more unique expressions than a single document, the resulting representation will only have few non-zero entries, hence the term "sparse representations" [52, 54]. Sometimes not the entire vocabulary is taken into consideration when constructing the embeddings, but only the most common terms across the training set are selected in order to reduce the sparsity[5] [55].

When raw word counts are used for encoding the texts, frequent words, such as stop words, can overshadow rare terms, which are often crucial for obtaining the correct prediction [54, 56]. To mitigate this issue, one can opt for *TF-IDF* (*Term Frequency - Inverse Document Frequency*) features, which put lower importance on words that appear in many texts across the corpus [52, 54]. The concrete formula for the TF-IDF value $f_{t,d}$ attributed to term t in document d from a collection of D documents is given by

$$f_{t,d} = tf_{t,d}\, idf_t$$

where

$$idf_t = \log(\frac{D}{df_t})$$

denotes the inverse document frequency of term t, $tf_{t,d}$ represents the number of occurrences of t in d, and df_t describes the number of documents in the collection that contain t (cf. Luan et al. 2021, Schütze et al. 2008, Subakti et al. 2022) [52, 54, 53]. As is evident from the above formula, $f_{t,d}$ is particularly large when term t appears frequently in d but only in very few documents across the collection. On the other hand, if the term t occurs in lots of texts in the corpus, idf_t will be very low, leading to an overall small TF-IDF value [54].

Despite TF-IDF features being widely used, especially for information retrieval, they have some disadvantages compared to more recent embedding methods [52, 54, 53]. Most notably, the individual TF-IDF values $f_{t,d}$ cannot take advantage of the context of term t [53].

[5] cf. the max_features parameter in the scikit-learn implementation `https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html`, last accessed 2022-06-23

Dense representations from pre-trained language models, on the other hand, can allow for context-dependent representations [53]. The BERT model, introduced by Devlin et al. (2018) [57], drastically improved performances on popular NLP benchmarks, such as GLUE [58], by learning *"bidirectional representations"* that depend on the context surrounding the word [57, 53]. Devlin et al. (2018) [57] pre-trained BERT on large amounts of unsupervised text, namely the BooksCorpus [59] and English Wikipedia, providing a good initialization for downstream tasks. Fine-tuning the model on task-specific data to further push performance is simple, since only an additional appropriate output layer is required to apply BERT to various NLP tasks [57]. For classification tasks, BERT provides a so-called *CLS token*, which encodes the entire sequence as a 768-dimensional vector [57, 53].

While the BERT model is very powerful, its large size can be impractical [57, 60]. Sanh et al. (2019) [60] therefore introduced DistilBERT, which is a downsized version of BERT that speeds up the fine-tuning process without substantially degrading the model performance.

In this thesis, both sparse TF-IDF features and dense representations, extracted from a fine-tuned DistilBERT model, are explored to assess the performance of the GM algorithm.

For the TF-IDF encodings, the maximum number of features was restricted to 1000, so only the 1000 most common terms in the training set received a dedicated representation. Dense representations were created by loading a pre-trained DistilBERT model from the *transformers* library [61] and fine-tuning it on the weakly supervised data[6]. In order to reduce the influence of randomness on the resulting embeddings, the model was only fine-tuned on training samples which had a definitive label assigned by majority voting, meaning 1844 instances for the SMS dataset and 1158 examples for the Youtube data (cf. Table 4.2) [7]. Due to the performance on the development set, the model was fine-tuned for one epoch on SMS and for two epochs on Youtube. After fine-tuning, the CLS representation was extracted for each data point.

4.3. Experimental Setup

In our experiments, the GM algorithm is only used to retrain the classification layer of the network. When DistilBERT embeddings are used, the parameters of all previous layers are frozen after fine-tunining. Since both datasets are binary classification tasks [7], the network has two output nodes; one corresponding to each class.

The output of the classification layer for sample t can be denoted by

$$z_{t,1} = \sum_{f=1}^{F} (w_{f,1}\, x_{t,f}) + b_1$$

[6]Fine-tuning was performed in PyTorch according to the tutorial `https://huggingface.co/transformers/v3.2.0/custom_datasets.html`, last accessed 2022-06-16

for the first output node, corresponding to class 0, and

$$z_{t,2} = \sum_{f=1}^{F} (w_{f,2}\, x_{t,f}) + b_2$$

for the second output node, representing class 1 [62]. Here, $x_{t,f}$ denotes the f-th feature of sample t and F represents the total number of features, which is 1000 for TF-IDF representations and 768 for DistilBERT embeddings. $w_{f,1}$ denotes the weight that connects the f-th input feature to the first output node and b_1 represents the bias term of the first output node. The terms $w_{f,2}$ and b_2 can be interpreted analogously.

In order to compute the loss, a softmax function [63] is applied to the two outputs. The resulting values will be abbreviated by

$$s_{t,1} = \frac{e^{z_{t,1}}}{e^{z_{t,1}} + e^{z_{t,2}}}$$

$$s_{t,2} = \frac{e^{z_{t,2}}}{e^{z_{t,1}} + e^{z_{t,2}}}$$

for the rest of the thesis. For the results presented in this chapter, the class with the higher softmax value was considered to be the model prediction.

In the following, the term *"baseline"* will be used to describe a model of the above architecture that was trained without any form of denoising.

The reported results for all configurations of the GM algorithm and the baseline are averaged over 10 runs. The training labels might slightly vary across the different runs, due to the ties being randomly broken. The same 10 seeds are used for the GM algorithm and the baseline, for both data preparation and training, to make the comparison as fair as possible. A grid search for the learning rate and the batch size, with the search space $\{0.01, 0.001, 0.0001\} \times \{32, 64, 128\}$, is performed on the development set in each individual run. The Adam optimizer [64] with default parameters[7], except for the learning rate, is used for updating the model. Each model is trained for 10 epochs and evaluated on the development set after each epoch. Once the 10 epochs are completed, the model that achieved the best evaluation score, i.e. F_1 for SMS and accuracy for Youtube [7], on the development set is loaded and evaluated on the test set.

4.3.1. Significance Testing

One of the primary goals of the conducted experiments is to determine whether models trained with the proposed denoising method can achieve better performances than the baseline. When comparing the results of two machine learning techniques, significance testing can be a useful tool to gauge whether the observed difference in performance is due to a better learning strategy or merely due to chance [65]. However, finding a suitable statistical test in the context of model comparison can be tricky, since assumptions

[7] cf. `https://pytorch.org/docs/stable/generated/torch.optim.Adam.html`, last accessed 2022-06-30

made by widely-used parametric tests are oftentimes not fulfilled [65]. As a result, non-parametric tests and randomization tests, which generally make fewer assumptions about the data, can be beneficial [66, 65].

Depending on the performance measure, different tests are recommended. When accuracy is used as the evaluation metric, an *exact binomial test* can give an indication whether one learning method truly outperforms the other, according to Salzberg (1997) [67]. The exact binomial test is a non-parametric test for dichotomous data, where 1 represents a success and 0 a failure. For the two-sided version of the test, the null hypothesis is that the true success probability p is equal to a pre-specified value p_0 [66, 67]. In order to apply this test in the context of model comparison, Salzberg (1997) [67] proposed the following procedure: First, the predictions of the two methods on the test $\mathcal{D}_E$ are dichotomized. A correct prediction is denoted by 1 and a false response is assigned the value 0. Keeping only the n predictions where one model succeeded while the other one failed, one can consider the number of times that system A was better than system B as the number of successes s. Under the null hypothesis, system A and system B are equally good. Therefore, one can test the null hypothesis $H_0 : p = 0.5$ on the basis of the n trials and s successes, in order to get an idea whether the performance difference between the two system is significant or not [67].

For the F_1 measure, Yeh (2000) [65] proposed using a randomization test[8] to assess whether the underlying models, that produced the results one wishes to compare, are really distinct from each other. The null hypothesis of this test is that there is no real difference between the two models [65].

The basic idea of randomization tests is the following [68, 65]: Given a test statistic computed on the original data, one can examine how often a random shuffling of datapoints between the groups achieves a test statistic that is at least as extreme as the original one. If this number is small, that is an indication that there is a real difference between the groups [68, 65].

The test proposed by Yeh (2000) [65] for the F_1 score goes as follows: Given the predictions of both systems on the test set, one can compute the initial absolute difference of the F_1 scores achieved by the two models and additionally check for which samples the two methods gave different results. Each prediction for which model A and model B disagree is flipped between the systems with a 50% chance. This results in a new prediction vector for both models. With these new predictions, the absolute difference in F_1 scores can be computed and compared to the difference obtained with the unaltered predictions. This process is repeated R times. Let r denote the number of times that the new absolute difference in F_1 scores is larger than or equal to the original difference. The maximum p-value is then given by $\frac{r+1}{R+1}$ [68, 65]. Note that R should be sufficiently large; Yeh (2000) [65] proposed $R = 1048576$. If the number of predictions that can be flipped is smaller than or equal to 20, Yeh (2000) [65] suggested trying all possible label reassignments (*exact randomization*). Pseudocode for the concrete testing procedure is given in Algorithm 3. For the tests conducted in this thesis, $\alpha = 0.05$ is chosen.

[8] An implementation of the test by Dmitry Ustalov can be found at `https://gist.github.com/dustalov/e6c3b9d3b5b83c81ecd92976e0281d6c`, last accessed 2022-06-16

Input : test predictions of system A: $\hat{y}_A = (\hat{y}_{A,1}, ..., \hat{y}_{A,|\mathcal{D}_E|})$,
test predictions of system B: $\hat{y}_B = (\hat{y}_{B,1}, ..., \hat{y}_{B,|\mathcal{D}_E|})$,
number of repetitions R
Output : p-value
Choose a significance level α;
Compute the initial difference in F_1 scores $t \leftarrow |F_1(\hat{y}_A) - F_1(\hat{y}_B)|$;
$\hat{y}^*_A \leftarrow \hat{y}_A$;
$\hat{y}^*_B \leftarrow \hat{y}_B$;
$r \leftarrow 0$;
for *j=1,...,R* **do**
  forall $\hat{y}_{A,i} \neq \hat{y}_{B,i}$ **do**
    With a 50% chance: $\hat{y}^*_{A,i} \leftarrow \hat{y}_{B,i}$ and $\hat{y}^*_{B,i} \leftarrow \hat{y}_{A,i}$
  end
  $t^* \leftarrow |F_1(\hat{y}^*_A) - F_1(\hat{y}^*_B)|$;
  if $t^* \geq t$ **then**
    $r \leftarrow r + 1$
  end
end
Compute the approximate p-value $\frac{r+1}{R+1}$

Algorithm 3: Randomization Test for F_1 Score Comparison by Yeh (2000)

4.3.2. Collected Statistics

Throughout each run, several statistics are collected to assess how well the different configurations of the denoising method can distinguish between correctly labeled and mislabeled samples. The following abbreviations will be used to describe these statistics.

Expression	Definition
CI (*Correctly Ignored*)	number of times a mislabeled sample seen during training was ignored
FI (*Falsely Ignored*)	number of times a correctly labeled sample seen during training was ignored
CK (*Correctly Kept*)	number of times a correctly labeled sample seen during training was kept
FK (*Falsely Kept*)	number of times a mislabeled sample seen during training was kept
CC (*Correctly Corrected*)	number of times a mislabeled sample seen during training was correctly relabeled
FC (*Falsely Corrected*)	number of times a correctly labeled sample seen during training was falsely relabeled

Table 4.3.: Collected Statistics for the GM Algorithm

Note that a given sample can be handled differently in each epoch. For example, a particular mislabeled sample could be ignored in the first epoch, increasing CI by 1, and kept in the second epoch, increasing FK by 1. Therefore, it holds that
$CI + FI + CK + FK + CC + FC = E^*|\mathcal{D}_T|$, where $|\mathcal{D}_T|$ is the size of the training set and E^* denotes the number of the epoch after which the model performed best on the development set. Remember that the model state after the completion of this epoch is loaded for the final evaluation on the test set. For simplicity, $E^*|\mathcal{D}_T|$ will be denoted by N for the rest of the thesis. According to the notation in Table 4.3, the total number of times

a sample was removed from the batch can be written as $I = CI + FI$, the total number of correctly labeled samples encountered during training is given by $CL = CK + FI + FC$ and the number of mislabeled samples that were seen during training can be computed by $FL = CI + FK + CC$. Note that if no alternative label is specified, it holds that $CC = 0$ and $FC = 0$.

The previously described expressions can be used to calculate some interesting characteristics of the training process, such as:

- $\frac{I}{N}$: fraction of times a sample seen during training was removed
- $\frac{CI}{I}$: fraction of times that the removal of a sample was warranted
- $\frac{CI}{FL}$: fraction of times a mislabeled sample seen during training was removed
- $\frac{FI}{CL}$: fraction of times a correctly labeled sample seen during training was removed
- $\frac{CC}{FC+CC}$: fraction of warranted label changes to the alternative label

Monitoring the above expressions for different settings of the GM algorithm can help us gauge how successful the method is in dynamically cleaning the data during training.

4.3.3. Examined Loss Functions

An advantage of the GM algorithm is that it can be used with a variety of loss functions, both for the gradient comparison and the actual update. Therefore, this section gives an overview of the different losses used in our experiments.

The cross-entropy (CE) loss[9] is a frequently used loss function, suitable for both binary and multi-class classification problems [69, 34]. In the binary setting, the averaged cross-entropy loss computed on batch $\mathcal{B}$ of size M can be written as [69, 34]:

$$\mathcal{L}_{CE}(\mathcal{B}) = \frac{1}{M} \sum_{t=1}^{M} l_t$$

where

$$l_t = -[\log(s_{t,2}) y_t + \log(s_{t,1})(1 - y_t)] = -[\log(s_{t,2}) y_t + \log(1 - s_{t,2})(1 - y_t)].$$

Despite the effectiveness of the cross-entropy loss in many scenarios, the loss has some downsides. First of all, there are some indications that the cross-entropy loss is suboptimal for learning with label noise, as was already mentioned in Section 2.4.1. Wang et al. (2019) [34] compared the performance of a classifier trained with the cross-entropy loss on the mostly clean CIFAR-10 dataset to a model trained in the same way on a version of CIFAR-10 with randomly perturbed labels. They found that the label noise noticeably impaired the model's score for several classes on the test set. Specifically, classes that are

[9] cf. `https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html` and `https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html`, last accessed 2022-06-16

generally considered challenging to predict seemed to suffer from the label flipping the most [34].

On another note, Bénédict et al. (2021) [19] point out that the cross-entropy loss does not directly represent the performance metric one might wish to optimize. However, directly maximizing evaluation metrics for classification tasks is tricky, since they usually require thresholding to convert the model outputs to predictions that can be used to compute the performance score [19]. One example of such a metric is the F_1 score, which is defined as the harmonic mean of precision and recall [70]. The F_1 score is frequently used to evaluate performance on imbalanced datasets, for which high accuracy can oftentimes be achieved by naively predicting the majority class for each sample [71]. Usually, the rare minority class samples are however of particular interest, which is why accuracy is a suboptimal performance metric for such datasets [72].

For binary classification, the concrete formula of the F_1 score of the positive class, computed on the dataset $\mathcal{D}$, is given by

$$F_1 = \frac{2tp}{2tp + fp + fn}$$

where

$$\begin{aligned} tp &= \sum_{t=1}^{|\mathcal{D}|} \mathbb{1}_{(s_{t,2} \geq T)} y_t \\ fp &= \sum_{t=1}^{|\mathcal{D}|} \mathbb{1}_{(s_{t,2} \geq T)} (1 - y_t) \\ fn &= \sum_{t=1}^{|\mathcal{D}|} \mathbb{1}_{(s_{t,2} < T)} y_t \end{aligned}$$

and T is the threshold that determines whether a sample is predicted to belong to the positive class or not (cf. Bénédict et al. 2021) [19, 73, 70]. Bénédict et al. (2021) [19] note that, due to the thresholding, the F_1 score is non-differentiable and therefore unsuited for gradient-based learning methods. In order to mitigate this problem, Bénédict et al. (2021) [19] developed a smooth version of the F_1 score by replacing the binary predictions in the F_1 formula by the predicted probabilities that are returned by the model. Their loss was originally developed for the multi-label case, so it uses a sigmoid activation function to transform the model outputs to probabilities and not a softmax [19]. However, exchanging the sigmoid with a softmax for the single-label case should not cause any problems. Therefore, the second loss function that is examined in the experiments is based on a soft version of the F_1 score of the positive class. In each batch $\mathcal{B}$ of size M, its components can be computed as follows (cf. Bénédict et al. 2021) [19].

$$\widehat{tp} = \sum_{t=1}^{M} s_{t,2} y_t$$
$$\widehat{fp} = \sum_{t=1}^{M} s_{t,2}(1 - y_t)$$
$$\widehat{fn} = \sum_{t=1}^{M} (1 - s_{t,2}) y_t$$

These expressions could be seen as the "expected" true positives, false positives and false negatives respectively. If the goal is to maximize the F_1 score of the positive class, the loss function

$$\mathcal{L}_{F_{1_{pos.}}}(\mathcal{B}) = 1 - \frac{2\widehat{tp}}{2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon}$$

could be used [19]. The parameter ϵ in the equation above denotes a small stabilizer that is added to the denominator of the loss term[10]. It should prevent the denominator of this loss from getting too close to zero, which can happen for small batches with extreme prediction values [74]. For the following experiments $\epsilon = 0.00001$ was chosen.

Unlike the cross-entropy loss, $\mathcal{L}_{F_{1_{pos.}}}$ cannot be decomposed into sample-wise contributions but is computed over the entire batch [19]. Note that this loss has a downside for highly imbalanced data: If no positively labeled sample is included in a particular batch, the weights will not be updated, since $\widehat{tp} = 0$, which causes an all-zero batch gradient (cf. A.1.1). Therefore, a large batch size is advised when using this loss [19]. A possible solution could be using a smooth version of the macro-F_1 instead, which averages the F_1 score of the positive and the negative class [70]. The soft macro-F_1 loss for the binary case takes the form

$$\mathcal{L}_{F_{1_{macro}}}(\mathcal{B}) = 1 - \frac{1}{2}\left[\frac{2\widehat{tp}}{2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon} + \frac{2\widehat{tn}}{2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon}\right]$$

where

$$\widehat{tn} = \sum_{t=1}^{M} (1 - s_{t,2})(1 - y_t)$$

as suggested by Bénédict et al. (2021) [19]. In contrast to $\mathcal{L}_{F_{1_{pos.}}}$, the gradient of $\mathcal{L}_{F_{1_{macro}}}$ does not automatically vanish if $\widehat{tp} = 0$. The conducted experiments will show whether any of these two F_1-based losses can increase performance compared to standard cross-entropy.

[10] Its inclusion was inspired by this implementation of a soft macro-F_1 loss for multi-label classification which can be found at `https://github.com/ashrefm/multi-label-soft-f1/blob/master/Multi-Label%20Image%20Classification%20in%20TensorFlow%202.0.ipynb`, last accessed 2022-06-29

4.3.4. Sampling Strategies for the Comparison Batch

Sampling the comparison batch is an important step in the GM algorithm. Therefore, two different sampling strategies are explored in the experiments.

The first sampling strategy simply draws random batches from the training data, without taking the label distribution of the particular dataset into account. This approach could potentially cause problems on datasets with high class imbalance. In particular, gradients of samples belonging to rare classes might not match the aggregated gradient computed almost exclusively on instances assigned to more common labels. As a result, minority class samples might be ignored or corrected by default, leading to a bad test performance.

Therefore, opting for class-weighted sampling might prove beneficial. For the experiments, the weight for class k will be computed as

$$v_k = \frac{1}{\sum_{t=1}^{|\mathcal{D}_T|} \mathbb{1}_{(y_t=k)}}$$

which is simply the inverted number of occurrences of this class in the training set (cf. Cao et al. 2019) [75]. As a result, samples of common classes are assigned a lower probability of being included into the comparison batch than instances of rare classes. The resulting comparison batch should therefore be representative of all classes. The outcomes of our experiments will indicate whether class-weighted comparison batch sampling can be beneficial or whether naive random sampling should be sufficient for most datasets.

4.4. Results

This section reports the performances of various settings of the GM algorithm on the SMS and the Youtube dataset, averaged over 10 runs. Especially interesting cases, with particularly low or high scores, will be examined in more detail to find out under which circumstances the algorithm performs well, or fails to do so, and why.

4.4.1. SMS

This segment summarizes the results achieved on the SMS dataset. Due to the large number of scenarios to investigate, regarding data preparation and the parameters of the denoising algorithm, the results and their discussion will be broken up into smaller sections.

As mentioned before, for all following experiments, samples without a single rule match were removed, reducing the size of the SMS training set to 1852 instances (cf. Table 4.2, Zhang et al. 2021) [76]. A comparison with the gold labels provided by WRENCH suggests that the average error rate of the weak labels across the 10 runs was around 2.94%.

Sparse Embeddings

Table 4.4 contains the results achieved on the SMS dataset by models trained with the GM algorithm using the cross-entropy loss and TF-IDF representations of the messages. All scores in columns whose headers contain "(Dev.)" are computed on the development set. The remaining columns contain the test performances. In order to keep the table concise, the removal threshold τ was set to 0 for all configurations. The effect of τ will be investigated in more detail in a later section. In accordance with the notation established in Table 3.1, the alternative label will be denoted by a. A slash (/) in the corresponding column will be used to signal that, in the respective configuration, no alternative label is used. In this case, the algorithm can only ignore samples, but not relabel them. The averaged performance of the baseline models is included as the last line of the table, separated from the other results by a horizontal line. As mentioned before, these baseline models were fit to the noisy data without any sort of correction. The standard deviations of the scores are reported in parentheses, next to the averaged performance metric.

Method	a	Weighted Sampling	Avg. F_1 (Dev.)	Avg. F_1	Avg. Precision	Avg. Recall
GM	/	No	0.84122(0.01822)	0.83353(0.00942)	0.96718(0.01924)	0.73284(0.0192)
GM	/	Yes	**0.90467(0.00613)**	**0.87567(0.01532)**	0.93356(0.02869)	**0.82537(0.02442)**
GM	0	No	0.79231(0.01873)	0.78973(0.03043)	0.94142(0.01947)	0.68209(0.04978)
GM	0	Yes	0.89381(0.00677)	0.86742(0.017)	0.92252(0.03445)	0.8194(0.02045)
Baseline	/	/	0.82025(0.00573)	0.81617(0.01583)	**0.9791(0.00068)**	0.7(0.02274)

Table 4.4.: Performance on SMS (TF-IDF, CE Loss, $\tau = 0$)

The baseline is outperformed, in terms of F_1 score, by all configurations of the GM algorithm listed in Table 4.4, except for the setting where an alternative label is included and the sampling of the comparison batch disregards the class imbalance. In particular, this selection of parameters results in a comparatively bad recall score for the positive class. On the contrary, not including an alternative label and using weighted sampling for the comparison batch results in a large performance gap over the baseline, in terms of F_1, due to much better recall. Overall, the most important factor for the performance of the GM algorithm in this scenario seems to be the sampling strategy. While specifying an alternative label is viable when weighted comparison batch sampling is used, it still degrades the performance for this specific dataset. It will be interesting to see whether these observations also hold for the dense representations as well as the Youtube dataset.

The next two sections take a closer look at the parameter combinations that resulted in the best and worst performance of the GM algorithm.

Worst GM Configuration The following paragraph aims to provide some intuition as to why the setting *"no weighted sampling, $a = 0$"* results in a large drop in performance, both compared to the baseline and other configurations of the denoising method.

The explicit formulas for the model gradients, computed with respect to the cross-entropy loss, give some insights into what might be the reason for the bad results. Let y_t denote the (weak) label of sample t and $x_{t,f}$ the f-th input feature of that sample. Moreover, let $s_{t,k}$ denote the value of the predicted probability that sample t belongs to

class k. The gradients of the individual model weights and biases with respect to sample t are then given by

$$\begin{aligned}
\frac{\partial l_t}{\partial w_{f,1}} &= (1 - s_{t,2})x_{t,f}y_t - s_{t,2}x_{t,f}(1 - y_t) \\
\frac{\partial l_t}{\partial w_{f,2}} &= (s_{t,2} - 1)x_{t,f}y_t + s_{t,2}x_{t,f}(1 - y_t) \\
\frac{\partial l_t}{\partial b_1} &= (1 - s_{t,2})y_t - s_{t,2}(1 - y_t) \\
\frac{\partial l_t}{\partial b_2} &= (s_{t,2} - 1)y_t + s_{t,2}(1 - y_t)
\end{aligned} \tag{4.1}$$

for the neural network used in the experiments (cf. A.1.1, Wang et al. 2019) [34]. Notice that the gradients corresponding to the weights and biases connected to output node 1 and output node 2 are exactly the same, except for a flipped sign. Moreover, when TF-IDF features are used, the inputs $x_{t,1}, ..., x_{t,F}$ are all non-negative [54]. As a result, if $y_t = 0$, it holds that $\frac{\partial l_t}{\partial w_{f,1}} \leq 0$ and $\frac{\partial l_t}{\partial b_1} \leq 0$, while $\frac{\partial l_t}{\partial w_{f,2}} \geq 0$ and $\frac{\partial l_t}{\partial b_2} \geq 0$. On the other hand, if $y_t = 1$, then $\frac{\partial l_t}{\partial w_{f,1}} \geq 0$ and $\frac{\partial l_t}{\partial b_1} \geq 0$, while $\frac{\partial l_t}{\partial w_{f,2}} \leq 0$ and $\frac{l_t}{\partial b_2} \leq 0$. Without class-weighted sampling, the comparison batch is likely to be dominated by instances with label 0, because roughly 86% of the samples in the SMS training set belong to this class (cf. Table 4.2) [17, 7]. Since gradients corresponding to samples with label 1 by default have a different orientation than gradients corresponding to samples from the majority class, there presumably is a high chance that the GM algorithm will correct them to label 0 in order to better match the comparison gradient. A look at the label changes during training, provided in Figure 4.1, affirms this intuition.

The left component of Figure 4.1 depicts the label changes performed by the GM algorithm before each batch update for an exemplary run of the setting *"no weighted sampling, $a = 0$, $\tau = 0$"*. Due to the results of the grid search that was performed for collecting the results in Table 4.4, learning rate 0.01 and batch size 32 was chosen for creating this plot. The x-axis represents the number of the update while the y-axis shows how the labels were changed before performing the corresponding optimization step. In particular, the depicted lines show the percentage of samples in the batch which were

- originally assigned label 0 and kept this way (blue line)
- originally assigned label 0 and ignored (orange line)
- originally assigned label 1 and kept this way (green line)
- originally assigned label 1 and changed to label 0 (red line)
- originally assigned label 1 and ignored (purple line)

As a result, for a given batch number, the corresponding values on the y-axis sum up to one. Note that only the label changes up to the chosen epoch E^* are depicted.

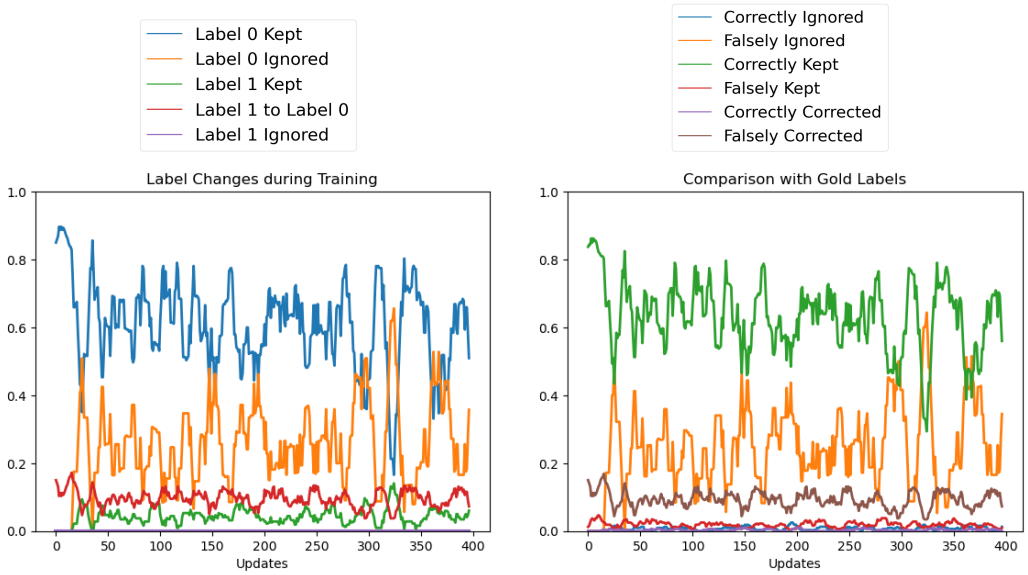


Figure 4.1.: Label Changes during Training for SMS
(TF-IDF, CE/CE, No Weighted Sampling, $a = 0$, $\tau = 0$)

In order to make the figure more easily readable, the statistics were smoothed over 10 batches, which is why the x-axis is slightly shortened.

Figure 4.1 confirms that most samples that were originally labeled as spam got reassigned to class 0 by the GM algorithm during training. On the other hand, the majority of instances labeled as ham by the weak supervision process are kept for most batch updates. Note that in this setting, positive labels cannot be ignored due to the gradient of the cross-entropy loss, given by the set of equations 4.1, and the formula chosen to compute the gradient similarity defined in Equation 3.1. If a sample with label 1 has a negative similarity score, changing its label to 0 will always result in a positive score, causing the sample to be relabeled for the update. As a result, positively labeled samples have to be either kept or reassigned to label 0.

Since the WRENCH benchmark provides gold labels for the SMS training set, it can be assessed whether the label changes made by the GM algorithm align with the ground-truth labels [7]. This comparison is provided in the right plot of Figure 4.1. The plot depicts the percentage of samples which were

- correctly ignored (blue line)
- falsely ignored (orange line)
- correctly kept (green line)
- falsely kept (red line)
- correctly corrected to the alternative label (purple line)

- falsely corrected to the alternative label (brown line)

For a given batch, the values depicted on the y-axis again sum up to one.

Judging from Figure 4.1, the vast majority of label changes from class 1 to class 0 were unwarranted. Looking at some statistics collected throughout the 10 runs, it turns out that, on average, only 3.83% of the label changes from class 1 to class 0 were actually correct. Moreover, the fraction of samples that were ignored despite being correctly labeled is concerningly high for many batches (cf. Figure 4.1). Across the 10 runs, around 24.70% of correctly labeled samples encountered during training were falsely removed from the batch. Lowering the removal threshold τ could potentially help mitigate this issue.

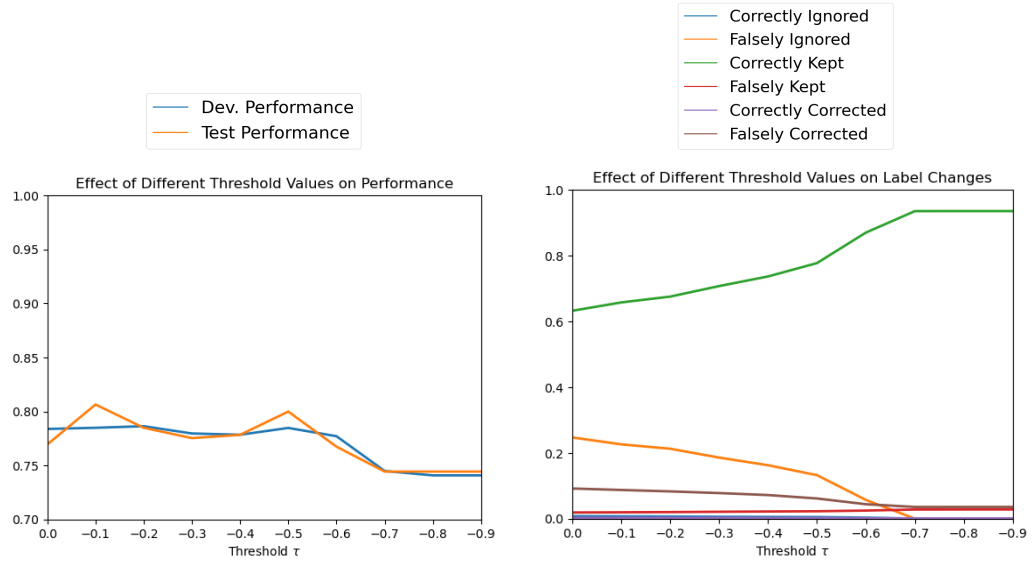


Figure 4.2.: Effect of Removal Threshold Value for SMS (TF-IDF, CE/CE, No Weighted Sampling, $a = 0$)

The left plot in Figure 4.2 demonstrates the effects of varying values of τ on the development and test performance while the right component gives insights into the corresponding label changes. The values in the plots are averaged over three runs of the GM algorithm for the specific threshold value. A grid search for the value of the learning rate and the batch size, with search space $\{0.01, 0.001, 0.0001\} \times \{32, 64, 128\}$, was performed in each run. The inspected thresholds are $\tau = 0, -0.1, -0.2, -0.3, -0.4, -0.5, -0.6, -0.7, -0.8, -0.9$.

As expected, the percentage of falsely ignored samples diminishes with decreasing values of τ and the portion of correctly kept samples rises. However, this comes at the cost of keeping more of the mislabeled samples. Since the value of the removal threshold naturally influences the dynamics of the label changes, it seems reasonable that the final performance is affected by the value of τ. Judging from Figure 4.2, $\tau \geq -0.6$ worked best in this scenario, but the resulting performance is not up to par with the other configurations, regardless of the threshold value.

Apparently, the high number of falsely ignored samples, as depicted in Figure 4.1, is not the primary cause for the bad performance of this setting. It seems likely that the unwarranted label changes, that are fairly prevalent at any threshold value, may be at fault. This seems reasonable, since the already small number of crucial minority class samples is further reduced.

Best GM Configuration The largest improvement over the baseline was achieved by adding no alternative label and using weighted sampling for the comparison batch. Figure 4.3 shows the label changes during an exemplary run of the GM algorithm with these parameters, learning rate 0.01, and batch size 64. The depicted statistics were not smoothed over several batches. The color coding of the lines is the same as in Figure 4.1.

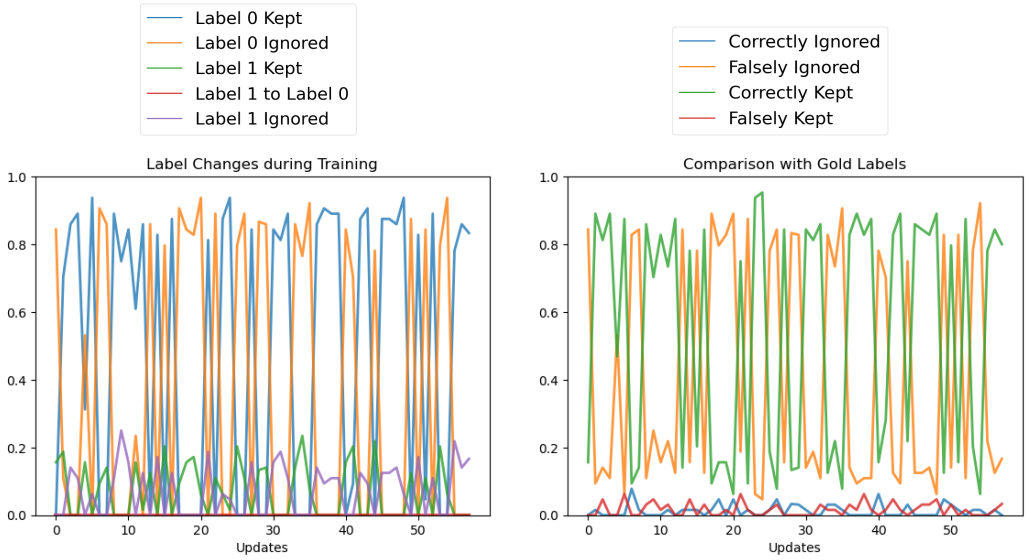


Figure 4.3.: Label Changes during Training for SMS
(TF-IDF, CE/CE, Weighted Sampling, No Alternative Label, $\tau = 0$)

The label dynamics for this scenario look fairly different from those presented in Figure 4.1. Surprisingly, in this configuration, a large portion of the minority class samples is ignored, despite the use of weighted sampling for the comparison batch. Moreover, several "pure" update batches, consisting either entirely of samples with weak labels 0 or 1, are created. A comparison with the gold labels can give some insight on how well this version of the GM algorithm can distinguish between mislabeled and correctly annotated samples. Interestingly, despite the improvement over the baseline, the share of falsely ignored samples per batch is quite high for many updates. Judging from Figure 4.3, the best-performing configuration of the GM algorithm found in Table 4.4 does not seem to be very effective in identifying mislabeled samples. The statistics collected throughout the 10 runs of the experiment reveal that, on average, only around 3.23% of ignored samples were actually mislabeled. Moreover, around 50.71% of the mislabeled instances

seen during training were ignored, while this percentage is only slightly lower for correctly labeled samples (45.95%).

Since a large fraction of correctly labeled samples is ignored in this scenario, decreasing the removal threshold τ might help to improve the performance further.

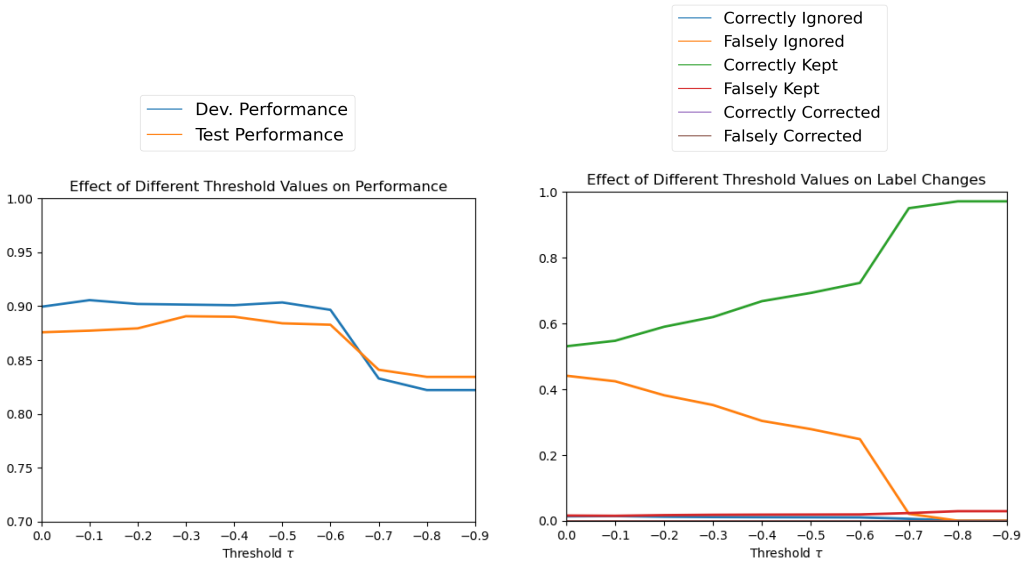


Figure 4.4.: Effect of Removal Threshold Value for SMS (TF-IDF, CE/CE, Weighted Sampling, No Alternative Label)

The left plot in Figure 4.4 shows that, for this combination of parameters, the value of τ seems to have a similar but slightly more significant effect than in the previously explored setting. In particular, choosing a threshold value lower than -0.6 noticeably degrades the performance for the depicted set of experiments. This is reflected in the right plot, as the jump from $\tau = -0.6$ to $\tau = -0.7$ has a very strong impact on the label dynamics. With decreasing removal threshold, this setting generally becomes more and more similar to the baseline, as less samples are ignored. Therefore, the score drop at low threshold values matches observation that the baseline performed worse than this particular configuration of the denoising method. If we focus purely on performance, choosing $\tau = 0$ seems sufficient in this case. Overall, the dynamics depicted in Figure 4.4 could indicate that keeping a larger portion of the mislabeled samples can be more harmful than ignoring large numbers of correctly labeled samples.

Significance of Results From all configurations examined in Table 4.4, the setting *"weighted comparison batch sampling, no alternative label"* achieved the best performance on the development set. Since the observed performance gap between this configuration of the GM algorithm and the baseline is fairly high, a randomization test at significance level $\alpha = 0.05$ will be conducted in order to assess whether there is a significant difference between the methods. Remember that the randomization test compares two particular

models and not the average performance scores reported in Table 4.4 (cf. Yeh 2000) [65]. Therefore, the best-scoring model on the development set found during the collection of Table 4.4 is retrieved for the baseline and the aforementioned GM configuration respectively. The number of disagreeing predictions of these two systems on the test set was smaller than 20, hence exact randomization was performed, as suggested by Yeh (2000) [65].

The test returned a p-value of 0.07213, which is higher than the significance level α. As a result, the null hypothesis, that there is no true performance difference between the two systems [65], is not rejected. This result can be seen as an indication that the performance gap observed in Table 4.4 might just be due to chance.

Now that that the results obtained by solely using the cross-entropy loss were analyzed, it will be investigated whether the performance of the baseline or the GM algorithm can be further pushed by making use of either $\mathcal{L}_{F_{1_{pos.}}}$ or $\mathcal{L}_{F_{1_{macro}}}$ as defined in Section 4.3.3.

First, the baseline performance using the two loss functions will be examined in order to assess whether additional experiments with these losses might be fruitful.

Method	Update Loss	Avg. F_1 (Dev.)	Avg. F_1	Avg. Precision	Avg. Recall
Baseline	F_1	**0.89189(0.01632)**	**0.87598(0.01447)**	0.97286(0.01212)	**0.79701(0.02355)**
Baseline	Macro-F_1	0.82986(0.01239)	0.82973(0.00591)	**0.98571(0.00986)**	0.71642(0.00704)
Baseline	CE	0.82025(0.00573)	0.81617(0.01583)	0.9791(0.00068)	0.7(0.02274)

Table 4.5.: Baseline Performance on SMS (TF-IDF, F_1-Based Losses)

The results listed in Table 4.5 show that using $\mathcal{L}_{F_{1_{pos.}}}$ to directly optimize the F_1 score of the positive class can indeed boost the performance. In particular, the recall score improves, while the precision ever so slightly decreases compared to the cross-entropy baseline. $\mathcal{L}_{F_{1_{macro}}}$ generally yields better results than the cross-entropy loss, however, the F_1 scores do not come close to the performance achieved with $\mathcal{L}_{F_{1_{pos.}}}$. This is somewhat intuitive, since $\mathcal{L}_{F_{1_{macro}}}$ optimizes the macro-F_1 score, while the chosen evaluation metric is the F_1 score of the positive class.

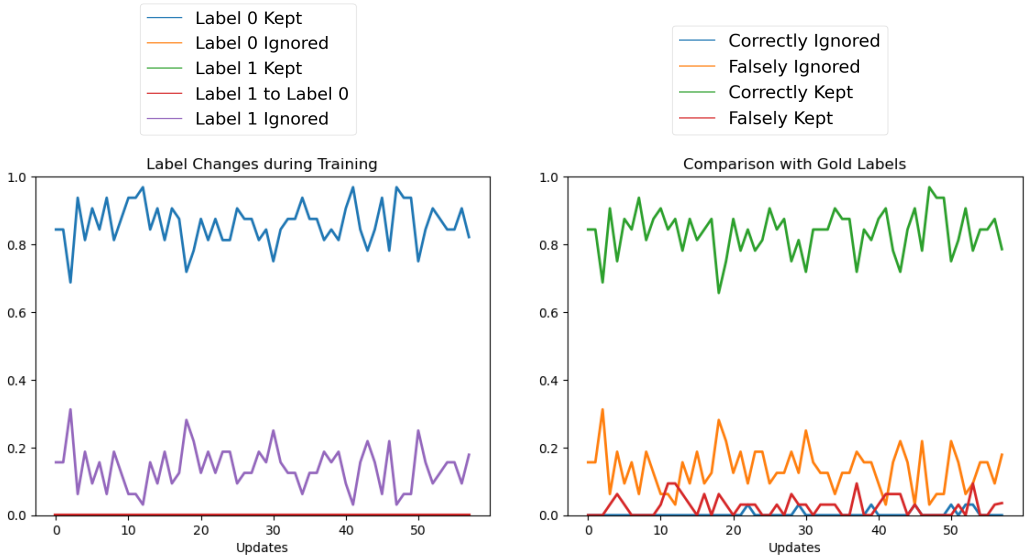
Table 4.6 summarizes the effects of the two F_1-based losses on the performance of the GM algorithm. Note that in the settings where an F_1-based loss is used as the comparison loss, no other class label is specified, since changing the label of one sample would affect the gradients of all other instances in the batch (cf. A.1.1).

Update/Comp. Loss	a	Weighted Sampling	Avg. F_1 (Dev.)	Avg. F_1	Avg. Precision	Avg. Recall
F_1/CE	/	No	0.19171(0.06908)	0.19712(0.08256)	0.1518(0.05382)	0.4403(0.25671)
F_1/CE	/	Yes	0.53371(0.14888)	0.54079(0.13555)	0.41122(0.14428)	**0.90597(0.14709)**
F_1/CE	0	No	0.19171(0.06908)	0.19712(0.08256)	0.1518(0.05382)	0.4403(0.25671)
F_1/CE	0	Yes	0.5139(0.16703)	0.51787(0.15805)	0.39677(0.16624)	0.8791(0.15818)
Macro-F_1/CE	/	No	0.86711(0.01355)	0.84997(0.01617)	0.97511(0.01263)	0.75373(0.02561)
Macro-F_1/CE	/	Yes	0.90731(0.00764)	0.88115(0.00968)	0.93261(0.02223)	0.83582(0.02225)
Macro-F_1/CE	0	No	0.85153(0.1157)	0.83752(0.01708)	0.95964(0.01375)	0.74328(0.02417)
Macro-F_1/CE	0	Yes	0.90398(0.00855)	0.8764(0.01403)	0.94846(0.02286)	0.81493(0.01888)
F_1/F_1	/	No	0.33299(0.06895)	0.34852(0.07875)	0.22837(0.06701)	0.89254(0.20786)
F_1/F_1	/	Yes	0.27641(0.03135)	0.27718(0.02719)	0.1785(0.0505)	0.80448(0.21066)
Macro-F_1/Macro-F_1	/	No	0.85074(0.0177)	0.84605(0.01367)	**0.98249(0.01083)**	0.74328(0.02312)
Macro-F_1/Macro-F_1	/	Yes	**0.91048(0.00928)**	**0.88249(0.00986)**	0.95082(0.02247)	0.82388(0.01835)

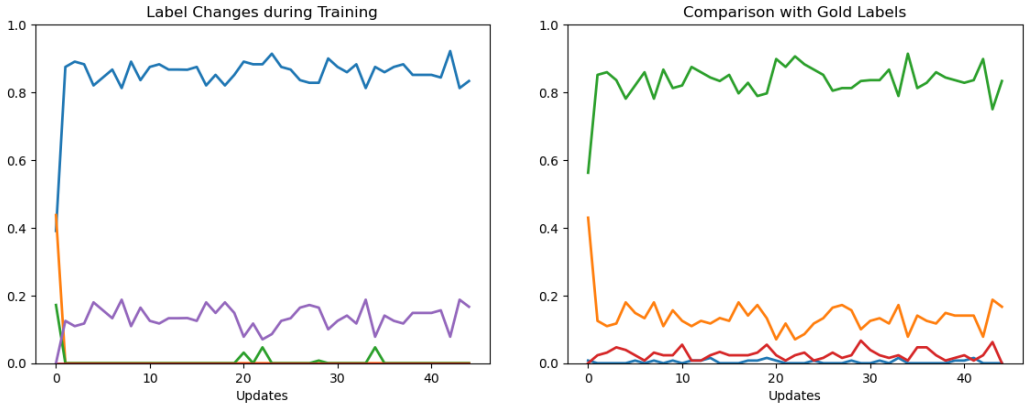
Table 4.6.: GM Performance on SMS (TF-IDF, F_1-Based Losses, $\tau = 0$)

Interestingly, updating with $\mathcal{L}_{F1_{pos.}}$ drastically deteriorates the performance of the GM algorithm. Using $\mathcal{L}_{F1_{macro}}$ for the update results in much better scores, even though the loss is based on a different evaluation metric. This is in contrast to the baseline results and likely has its roots in the label changes induced by the denoising algorithm. The following section aims to find a possible explanation for the poor performance of $\mathcal{L}_{F1_{pos.}}$ as an update loss for the GM algorithm.

Worst GM Configuration The worst average F_1 score recorded in Table 4.6 was obtained by using $\mathcal{L}_{F1_{pos.}}$ to perform the update and $\mathcal{L}_{CE}$ to compute the compared gradients. Using weighted sampling for the comparison batch improves the results for this loss combination, however, the final average score is still much lower than the one obtained by updating with the cross-entropy loss. Figure 4.5 gives insights into the label changes induced by this loss combination when no alternative label is specified. Learning rate 0.01 and batch size 32 was selected for the run with uniform comparison batch sampling. For the setting that makes use of weighted comparison batch sampling, learning rate 0.01 and batch size 128 was chosen by the grid search. In both cases, the statistics were not smoothed over several batches, since the plots were already nicely readable.



(a) F_1/CE, No Weighted Sampling, No Alternative Label, LR=0.01, $M = 32$



(b) F_1/CE, Weighted Sampling, No Alternative Label, LR=0.01, $M = 128$

Figure 4.5.: Label Changes during Training for SMS (TF-IDF, F_1/CE, $\tau = 0$)

Figure 4.5 (a) reveals that, without weighted comparison batch sampling, all minority class instances are ignored, at least in the depicted run. As a result, no updates are conducted, since when using $\mathcal{L}_{F1_{pos.}}$, all gradient entries are zero when the expected number of true positives in the update batch is zero (cf. A.1.1). Therefore, the resulting system is still equal to the randomly initialized model, which explains its poor performance.

Modifying the removal threshold could potentially help mitigate this issue. Figure 4.6 examines whether this intuition is correct.

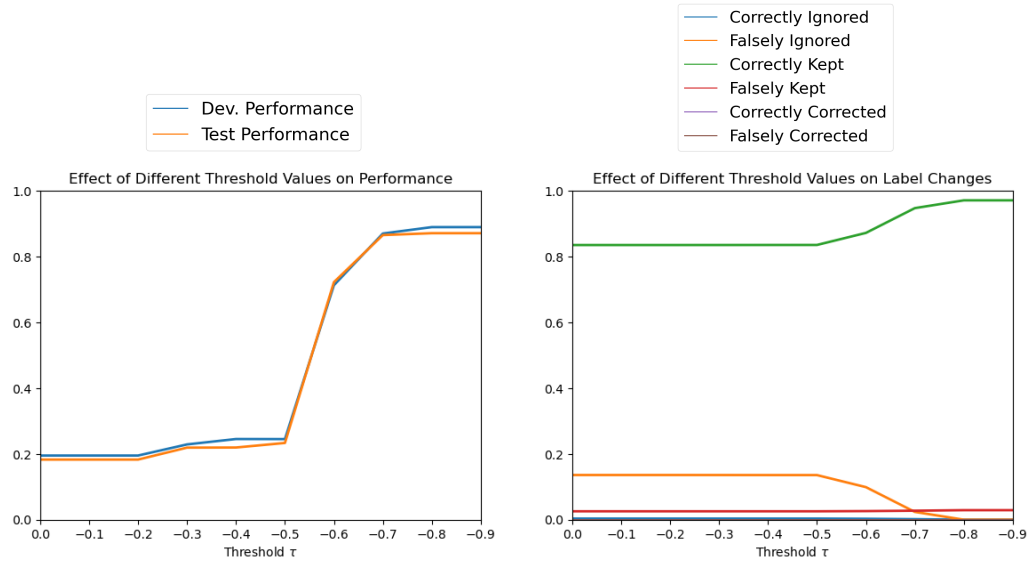


Figure 4.6.: Effect of Removal Threshold Value for SMS
(TF-IDF, F_1/CE, No Weighted Sampling, No Alternative Label)

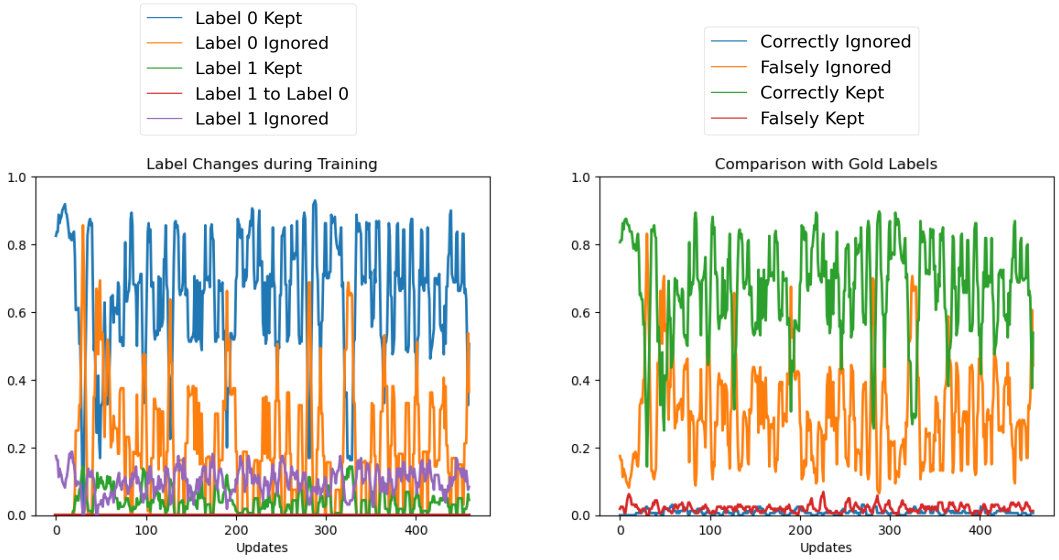
As expected, decreasing the threshold τ "forces" the algorithm to keep some minority class samples, despite them having a negative similarity score with the comparison batch. Performance rapidly increases when $\tau \leq -0.6$ is chosen. This result shows that the "default" removal threshold, $\tau = 0$, is not always optimal, and therefore, tuning the value of τ can prove beneficial.

Figure 4.5 (b) indicates that, surprisingly, almost all minority class samples are ignored by the denoising method, even when weighted comparison batch sampling is used. This behaviour of the GM algorithm is highly undesirable, since simply ignoring the minority class is not an effective way to diminish label noise. At first glance, it is a bit perplexing that the label dynamics depicted in Figure 4.5 (b) are so vastly different from those observed when the basic cross-entropy loss was to update (cf. Figure 4.3). After all, the only difference between the two settings is the update loss, while the formulas for computing the comparison gradient and the individual sample gradients are the same. However, it is important to remember that the predicted probabilities of the current model directly influence the gradients and by extension the similarity score of each sample. Even with weighted comparison batch sampling, a model trained with $\mathcal{L}_{F1_{pos.}}$ rarely keeps positive samples, and therefore seldomly updates (cf. Figure 4.3). Therefore, it likely produces very different outputs than a model that is "properly" trained with the cross-entropy loss, which possibly explains the observed difference in the label dynamics.

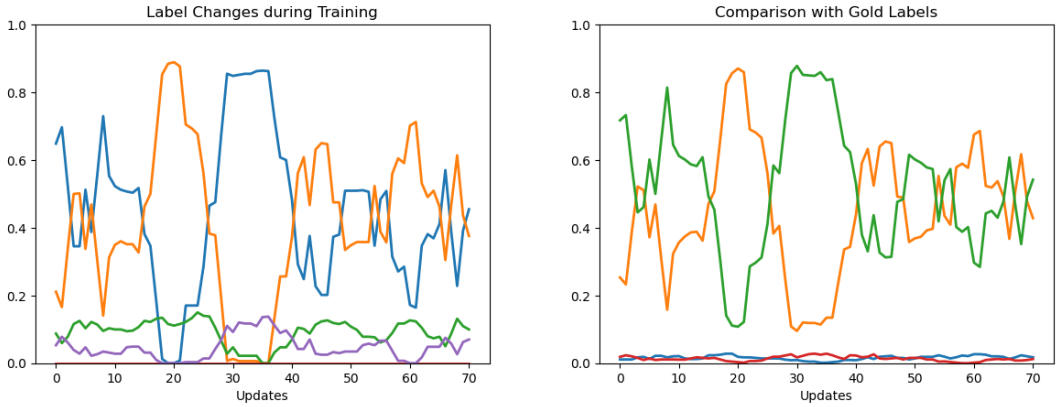
In conclusion, training with $\mathcal{L}_{F1_{pos.}}$ should be handled with care when the GM algorithm is used for denoising, since it cannot be guaranteed that there will be positive samples in the resulting update batch. Tuning the removal threshold is a possible solution to this problem, however, it might be more efficient to simply choose a different loss instead. For example, updating with $\mathcal{L}_{F1_{macro}}$ worked well, even for $\tau = 0$, according to the results in

Table 4.6. The following segment therefore explores the effects of training with $\mathcal{L}_{F_{1_{macro}}}$.

Best GM Configuration Figure 4.7 presents analogous plots to those provided in Figure 4.5 for the the setting where $\mathcal{L}_{F_{1_{macro}}}$ is chosen as both the update loss and the comparison loss. This loss assignment achieved the best development and test performance in Table 4.6 when combined with weighted sampling for the comparison batch. For better readability, the depicted statistics in Figure 4.7 (a) and (b) were smoothed over five batches.



(a) Macro-F_1/Macro$-F_1$, No Weighted Sampling, No Alternative Label, LR=0.01, $M = 32$



(b) Macro-F_1/Macro$-F_1$, Weighted Sampling, No Alternative Label, LR=0.01, $M = 128$

Figure 4.7.: Label Changes during Training for SMS (TF-IDF, Macro-F_1/Macro-F_1, $\tau = 0$)

The training dynamics of the two scenarios depicted in Figure 4.7 seem fairly similar for

the first few updates, however, the model utilizing weighted comparison batch sampling reached its best development performance in a much earlier epoch than the other configuration. Despite the good results achieved by using the soft macro-F_1 loss for both the gradient comparison and the update, the label dynamics still do not suggest that the method is effectively distinguishing between mislabeled and correctly labeled samples. Judging from the statistics collected over 10 runs, the average fraction of mislabeled samples that are ignored is around 54.76% when weighted sampling for the comparison batch is used (cf. Figure 4.7 (b)). For correctly labeled samples, the percentage is only slightly lower at 48.07%.

To summarize, the experiments conducted on sparse TF-IDF representations of the SMS dataset showed that applying Algorithm 1 can ameliorate performance compared to training on unaltered noisy labels. However, the resulting difference in test scores was not significant for the setting with the largest performance gap. Moreover, the performance of the GM algorithm highly depends on the chosen loss functions and parameters. In particular, using weighted sampling for the comparison batch seems to be crucial for achieving a high F_1 score with the denoising method on this specific dataset. Comparing the label changes conducted by the GM algorithm with the ground-truth labels provided by WRENCH showed no indication that the method can discern between mislabeled and correctly labeled samples, at least for TF-IDF encodings of the input texts. The next section explores whether the observations outlined above also hold when dense embeddings are used.

Dense Embeddings

In this segment, the previous experiments are recreated with embeddings extracted from a fine-tuned DistilBERT model. The main differences to the previous experiments with TF-IDF encodings are that the features can also be negative and that the gradients now generally have a larger number of non-zero entries.

Table 4.7 summarizes the results obtained by solely using the cross-entropy loss, akin to Table 4.4.

Method	a	Weighted Sampling	Avg. F_1 (Dev.)	Avg. F_1	Avg. Precision	Avg. Recall
GM	/	No	**0.91817(0.00656)**	**0.91328(0.02093)**	0.95444(0.00691)	**0.87612(0.03659)**
GM	/	Yes	0.9136(0.00527)	0.89912(0.01064)	0.9517(0.00496)	0.85224(0.0192)
GM	0	No	0.91778(0.00513)	0.91147(0.01968)	**0.95719(0.0118)**	0.87015(0.02819)
GM	0	Yes	0.91494(0.00415)	0.90086(0.00933)	0.95187(0.00487)	0.85522(0.01731)
Baseline	/	/	0.91469(0.00321)	0.89396(0.006)	0.95121(0.00506)	0.84328(0.01055)

Table 4.7.: Performance on SMS (DistilBERT, CE Loss, $\tau = 0$)

The performances in Table 4.7 show that training on embeddings extracted from DistilBERT generally results in better F_1 scores than using simple TF-IDF features (cf. Table 4.4). The recall is however still much lower than the precision. The baseline is consistently outperformed by the GM algorithm, in terms of average test F_1 score. However, the performance gap is much smaller compared to the results listed in Table 4.4.

Moreover, looking at the standard deviations, the test F_1 scores achieved by models trained with the GM algorithm are more volatile than the corresponding baseline performances.

Since the results are very close and the standard deviations are fairly high, picking a designated "best" and "worst" parameter combination for this set of experiments does not seem sensible. However, it is interesting to see that the effects of the parameters on the final performance are vastly different in this setting compared to the experiments conducted on sparse features (cf. Table 4.4). Surprisingly, making use of weighted sampling for the comparison batch did not seem to affect the final performance much, despite it being crucial in the TF-IDF setting. Furthermore, for sparse features, setting $a = 0$ and using uniformly sampled comparison batches resulted in a considerable performance drop compared to the baseline. However, this does not seem to be the case when DistilBERT embeddings are used.

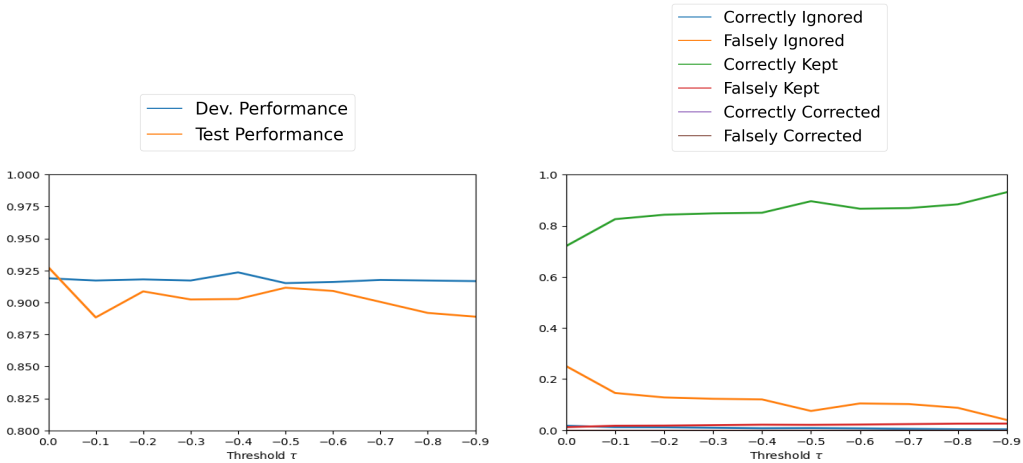
Comparing the label changes conducted over the 10 runs with the ground-truth labels reveals that the proposed denoising method seems to be more capable of distinguishing between mislabeled and correctly labeled samples when dense features are used. This could be due to a better representation of the samples compared to TF-IDF features, or the fact, that the sample-wise gradients are not sparse anymore. Moreover, the possibility of having negative features could play into this observation. The following table summarizes the statistics collected throughout the 10 runs for each configuration of the GM algorithm listed in Table 4.7. The notation introduced in Section 4.3.2 (cf. Table 4.3) was used for creating the column names.

a	Weighted Sampling	$\frac{I}{N}$	$\frac{CI}{FL}$	$\frac{FI}{CL}$	$\frac{CC}{CC+FC}$
/	No	0.24211(0.06421)	0.5709(0.04293)	0.23218(0.06522)	/
/	Yes	0.31888(0.17403)	0.70844(0.04682)	0.30713(0.17818)	/
0	No	0.17779(0.04339)	0.60751(0.03493)	0.16478(0.04483)	0.10698(0.02117)
0	Yes	0.24408(0.12023)	0.68904(0.04844)	0.23065(0.12242)	0.21198(0.16726)

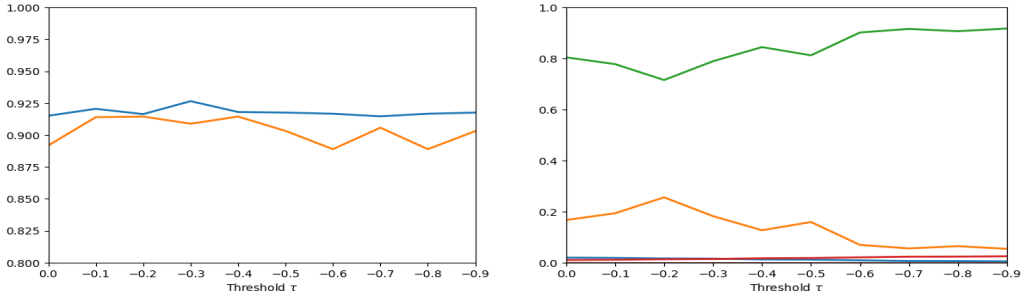
Table 4.8.: Statistics of Label Changes on SMS (DistilBERT, CE Loss, $\tau = 0$)

Judging from Table 4.8, the GM algorithm generally ignored less samples than in the TF-IDF setting. The average fraction of correctly labeled samples that are ignored during training ($\frac{FI}{CL}$) is however still fairly high for each parameter combination. Nevertheless, it is significantly lower than the percentage of mislabeled samples that are removed ($\frac{CI}{FL}$), which indicates that the method is potentially able to discriminate between correct and noisy samples with this encoding. However, when allowing for an alternative label, the fraction of correct label changes ($\frac{CC}{CC+FC}$) is still fairly low, especially without weighted comparison batch sampling.

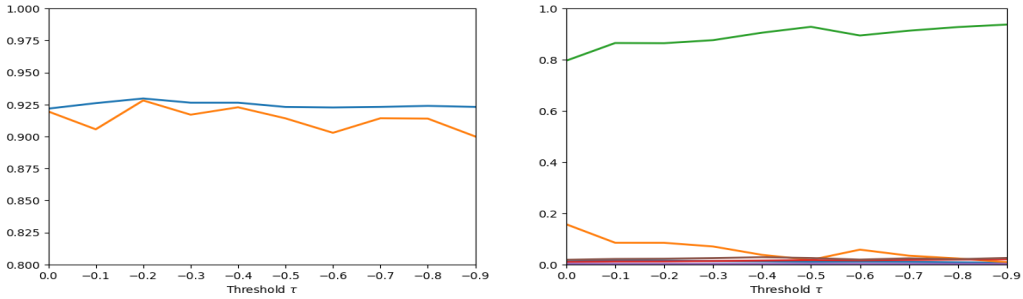
In the following, the effect of τ on the four scenarios presented in Table 4.7 is analyzed, to see if tuning the removal threshold could improve the performance further.



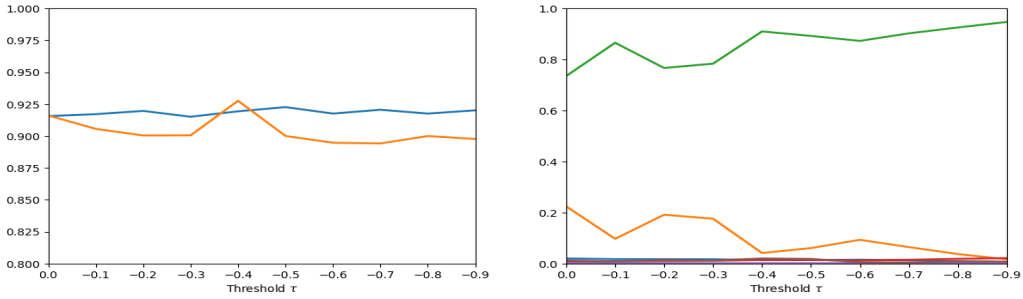
(a) No Weighted Sampling, No Alternative Label



(b) Weighted Sampling, No Alternative Label



(c) No Weighted Sampling, $a = 0$



(d) Weighted Sampling, $a = 0$

Figure 4.8.: Effect of Removal Threshold Value for SMS (DistilBERT, CE/CE)

Figure 4.8 shows that the value of the removal threshold does not seem to strongly affect the performance on the development set. At first glance, the harsh "spikes" in the label dynamics in Figure 4.8 (b) and (d) seem odd, as one would expect a more smooth decay of the fraction of falsely ignored samples per batch as τ decreases. However, a look at the non-averaged statistics reveal that even for a given threshold, the dynamics can vary greatly depending on the seed, especially when weighted comparison batch sampling is used. This observation, combined with the fact that only three runs per threshold were averaged for creating Figure 4.8, could explain the unstable behaviour. For all scenarios depicted in Figure 4.8, choosing $\tau = 0$ seems sufficient.

Next, the effects of using F_1-based losses with dense representations for the SMS dataset are explored.

First, akin to Table 4.5, the performance of the baseline models trained with $\mathcal{L}_{F_{1_{pos.}}}$ and $\mathcal{L}_{F_{1_{macro}}}$ is assessed in Table 4.9.

Method	Update Loss	Avg. F_1 (Dev.)	Avg. F_1	Avg. Precision	Avg. Recall
Baseline	F_1	0.90686(0.00498)	0.90088(0.01059)	**0.95179(0.00547)**	0.85522(0.01581)
Baseline	Macro-F_1	0.90486(0.00657)	**0.90785(0.01129)**	0.95096(0.00107)	**0.86866(0.01965)**
Baseline	CE	**0.9136(0.00527)**	0.89396(0.006)	0.95121(0.00506)	0.84328(0.01055)

Table 4.9.: Baseline Performance on SMS (DistilBERT, F_1-Based Losses)

Surprisingly, neither F_1 loss makes a large impact on the final performance. In fact, on the development set, the scores are even slightly worse compared to the cross-entropy loss. Nevertheless, it will be interesting to see whether updating with $\mathcal{L}_{F_{1_{pos.}}}$ also degrades the performance of models trained with the GM algorithm when dense embeddings are used.

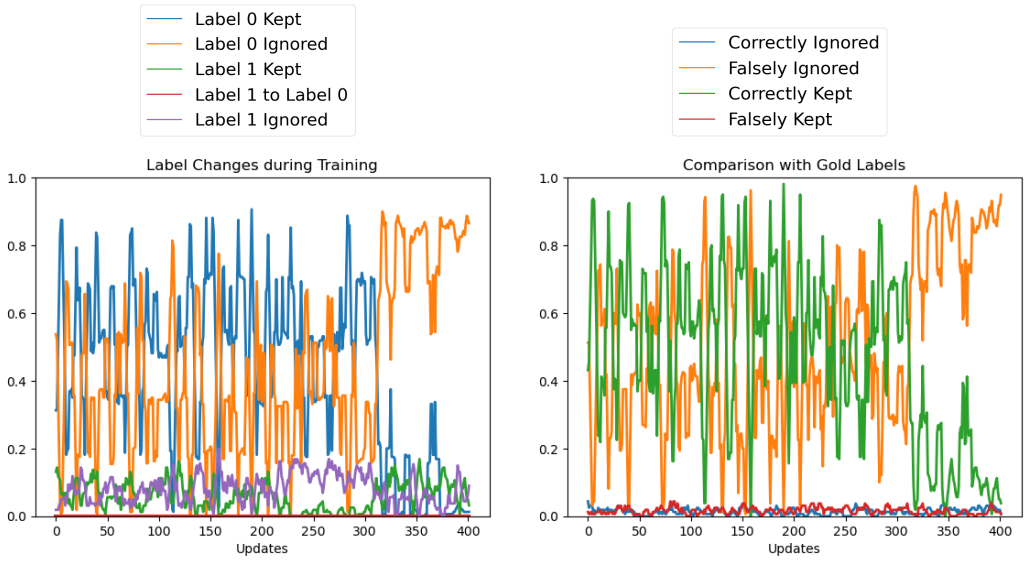
Table 4.10 summarizes the results for the GM algorithm applied to dense embeddings with various combinations of loss functions.

Update/Comp. Loss	a	Weighted Sampling	Avg. F_1 (Dev.)	Avg. F_1	Avg. Precision	Avg. Recall
F_1/CE	/	No	0.89933(0.00758)	0.92714(0.01888)	0.9488(0.00849)	0.90746(0.0403)
F_1/CE	/	Yes	0.89798(0.00708)	**0.94143(0.01112)**	0.94472(0.0087)	**0.93881(0.02854)**
F_1/CE	0	No	0.89961(0.00793)	0.92522(0.01922)	0.95158(0.00925)	0.90149(0.04291)
F_1/CE	0	Yes	0.8986(0.0071)	0.9349(0.01345)	0.94543(0.00894)	0.92537(0.03224)
Macro-F_1/CE	/	No	**0.91591(0.00204)**	0.89464(0.0081)	0.95284(0.00673)	0.84328(0.0145)
Macro-F_1/CE	/	Yes	0.90806(0.00474)	0.90359(0.01072)	0.95056(0.00102)	0.86119(0.01868)
Macro-F_1/CE	0	No	0.91135(0.00366)	0.89884(0.00875)	**0.95487(0.01087)**	0.84925(0.01643)
Macro-F_1/CE	0	Yes	0.90836(0.00537)	0.89995(0.01222)	0.95178(0.00496)	0.85373(0.02203)
F_1/F_1	/	No	0.89846(0.00695)	0.93312(0.01191)	0.948(0.00893)	0.9194(0.03002)
F_1/F_1	/	Yes	0.89996(0.00759)	0.9289(0.02088)	0.94623(0.00877)	0.91343(0.04439)
Macro-F_1/Macro-F_1	/	No	0.89928(0.00884)	0.9292(0.0121)	0.94609(0.00633)	0.91343(0.02797)
Macro-F_1/Macro-F_1	/	Yes	0.90014(0.00802)	0.92487(0.01952)	0.94572(0.00558)	0.90597(0.04165)

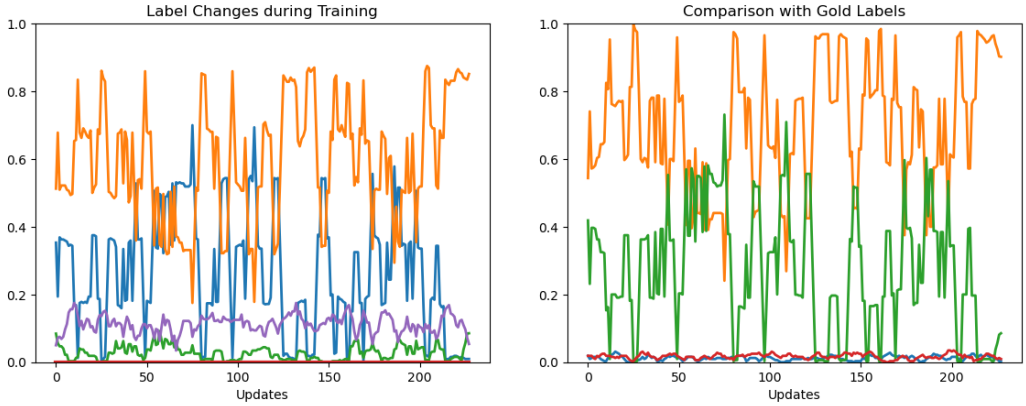
Table 4.10.: Test Performance on SMS (DistilBERT and F_1-Based Losses)

The different settings do not affect the performance as drastically as when sparse encodings were used (cf. Table 4.6). Interestingly, choosing $\mathcal{L}_{F_{1_{pos.}}}$ as the update loss and $\mathcal{L}_{CE}$ as the comparison loss turns out to be the best-performing configuration on the test set when DistilBERT embeddings are used, due to a comparatively high recall score. When the data was encoded with TF-IDF, this setting ignored almost all minority class

samples, even when weighted sampling for the comparison batch was used (cf. Figure 4.5), and therefore resulted in a very bad performance (cf. Table 4.10). Figure 4.9 reveals that this is not the case anymore when dense embeddings are chosen. Even without class-weighted comparison batch sampling, at least some positive samples are kept per batch, causing the model to properly update. However, the high fraction of falsely ignored samples is concerning. It is very interesting that the performance is not negatively affected by the removal of this many data points, especially considering the small size of the training set.



(a) F_1/CE, No Weighted Sampling, No Alternative Label, LR=0.01, $M = 32$



(b) F_1/CE, Weighted Sampling, No Alternative Label, LR=0.01, $M = 64$

Figure 4.9.: Label Changes during Training for SMS (DistilBERT, F_1/CE, No Alternative Label, $\tau = 0$)

While the results on the SMS dataset already gave some valuable insights into the

potential benefits and shortcomings of the proposed denoising method, evaluating its performance on only one dataset can be misleading. Therefore, the following segment repeats the experiments conducted on the SMS data on the Youtube dataset. The main difference between these two data collections is that the Youtube dataset does not have a strong class imbalance, unlike the SMS data. Note that the results for the Youtube data therefore will be measured in accuracy instead of F_1, following Zhang et al. (2021) [7].

4.4.2. Youtube

Akin to the experiments on the SMS dataset, examples which were not covered by any of the labeling functions were removed from the dataset. This design choice reduces the size of the Youtube training set to 1391 instances (cf. Table 4.2, Zhang et al. 2021) [7]. The average noise rate of the training labels across the 10 runs was around 13.85%, which is much higher than the fraction of labeling errors in the SMS data. An important factor for the comparatively large amount of incorrect annotations is the high number of ties in the Youtube dataset: 233 out of 1391 samples received the same number of votes for label 0 and 1 and are therefore assigned random labels in the data preparation phase. Removing the ties reduces the noise rate to 6.65%. For the experiments presented in this theses, the ties are however kept in the dataset, since it will be interesting to see how well the GM algorithm performs in comparison to the baseline on data with substantial label noise.

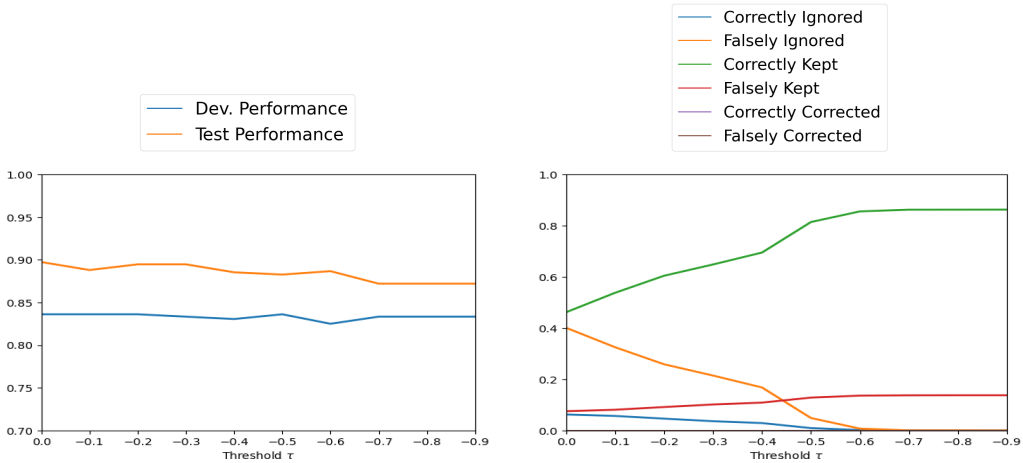
Sparse Embeddings

Table 4.11 was collected equivalently to Table 4.4.

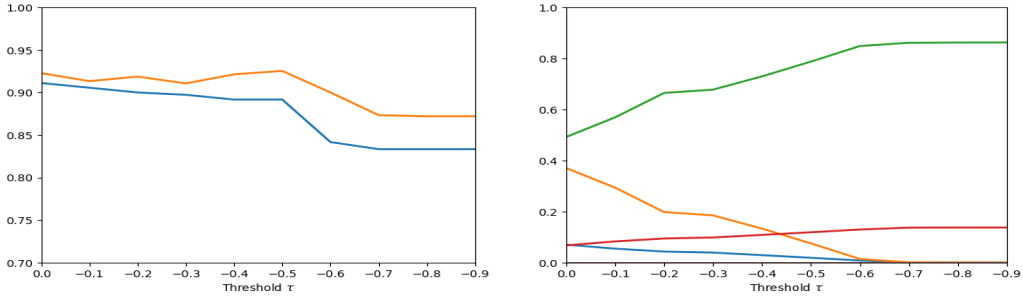
Method	a	Weighted Sampling	Avg. Accuracy (Dev.)	Avg. Accuracy
GM	/	No	0.8425(0.02952)	0.8936(0.01635)
GM	/	Yes	**0.89333(0.01459)**	**0.9156(0.00479)**
GM	0	No	0.84(0.01703)	0.8988(0.01677)
GM	0	Yes	0.8575(0.0127)	0.898(0.02446)
Baseline	/	/	0.83417(0.02132)	0.8804(0.01918)

Table 4.11.: Performance on Youtube (TF-IDF, CE Loss, $\tau = 0$)

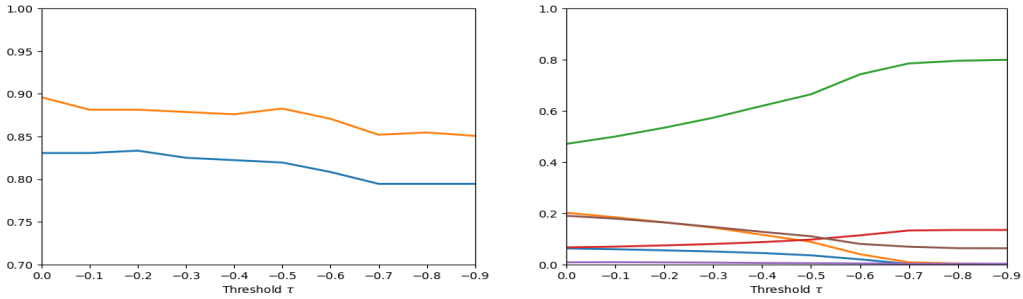
Table 4.11 summarizes results obtained by training with the cross-entropy loss, averaged over 10 runs. Every configuration of the denoising method outperforms the baseline. This may be due to the higher noise rate in the dataset or the absence of strong class imbalance which (presumably) ran some parameter configurations uncompetitive on the SMS data. However, the test performance gap between the baseline and the best configuration of the denoising method is not as high as on the SMS dataset (cf. Table 4.4). Figure 4.10 examines whether the above results can be further improved by decreasing the removal threshold τ.



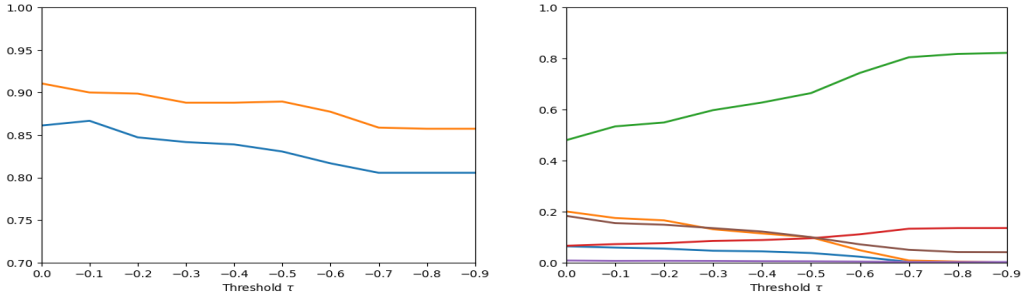
(a) No Weighted Sampling, No Alternative Label



(b) Weighted Sampling, No Alternative Label



(c) No Weighted Sampling, $a = 0$



(d) Weighted Sampling, $a = 0$

Figure 4.10.: Effect of Removal Threshold Value for Youtube (TF-IDF, CE/CE)

Generally, the influence of τ seems to depend on the setting. When no alternative label is specified and uniform comparison batch sampling is chosen, the value of τ barely influences the performance, despite the corresponding label dynamics changing drastically. For the remaining settings, there seems to be a downwards trend in performance as τ decreases. Subfigures (c) and (d), which correspond to the settings incorporating an alternative label, reveal that the fraction of falsely corrected samples is fairly high, even when weighted comparison batch sampling is used. It is interesting that these incorrect label reassignments do not seem to hurt the test performance much compared to the settings without an alternative label, namely (a) and (b).

Significance of Results Considering all of the results collected in Table 4.11 , the GM setting *"weighted sampling, no alternative label"* achieved the best development performance across the 10 runs with a mean accuracy of around 0.89333. An exact binomial test, as described by Salzberg (1997) [67], is conducted to determine whether the resulting improvement over the baseline on the test set is significant. The test compares the test predictions of the best-performing models, with respect to the development accuracy, found for the baseline and the GM algorithm. The significance level is chosen as $\alpha = 0.05$. The test resulted in a p-value of 0.3323, which is higher than the significance level. Therefore, the null hypothesis is not rejected. The observed performance difference between the two models might be due to chance.

While the performance measure for this dataset is accuracy and not F_1, it could still prove beneficial to examine how well $\mathcal{L}_{F_{1_{pos.}}}$ and $\mathcal{L}_{F_{1_{macro}}}$ perform on this dataset. In particular, it will be interesting to see whether the main observation from Tables 4.5 and 4.6, namely that $\mathcal{L}_{F_{1_{pos.}}}$ works well as an update loss for the baseline but not for the GM algorithm in the case of TF-IDF features, also holds for the more balanced Youtube data.

Method	Update/Comp. Loss	a	Weighted Sampling	Avg. Accuracy (Dev.)	Avg. Accuracy
GM	F_1/CE	/	No	0.74667(0.0448)	0.7248(0.05596)
GM	F_1/CE	/	Yes	0.71833(0.06768)	0.7052(0.0604)
GM	F_1/CE	0	No	0.71917(0.0557)	0.6924(0.05398)
GM	F_1/CE	0	Yes	0.70917(0.07822)	0.6936(0.06856)
GM	Macro-F_1/CE	/	No	0.83667(0.02612)	0.8984(0.1103)
GM	Macro-F_1/CE	/	Yes	0.88417(0.01942)	0.9104(0.00965)
GM	Macro-F_1/CE	0	No	0.82417(0.02272)	0.8796(0.02162)
GM	Macro-F_1/CE	0	Yes	0.85333(0.01532)	0.8988(0.00844)
GM	F_1/F_1	/	No	0.60333(0.05882)	0.5876(0.05162)
GM	F_1/F_1	/	Yes	0.6025(0.05919)	0.592(0.04688)
GM	Macro-F_1/Macro-F_1	/	No	0.835(0.02772)	0.8904(0.01388)
GM	Macro-F_1/Macro-F_1	/	Yes	**0.89083(0.01732)**	0.9124(0.00717)
Baseline	F_1	/	/	0.8875(0.00708)	**0.9372(0.00755)**
Baseline	Macro-F_1	/	/	0.845(0.002194)	0.8916(0.01712)
Baseline	CE	/	/	0.83417(0.02132)	0.8804(0.01918)

Table 4.12.: Performance on Youtube (TF-IDF, F_1-Based Losses, $\tau = 0$)

Table 4.12 affirms the observation that $\mathcal{L}_{F1_{pos.}}$ performs poorly as an update loss for the GM algorithm, at least for sparse encodings. Its negative effect is however not as drastic as on the SMS data, which is likely due to the more balanced class distribution in the training set. Interestingly, $\mathcal{L}_{F1_{pos.}}$ noticeably boosts the performance of the baseline models, compared to the standard cross-entropy loss, despite optimizing the F_1 score of the positive class and not the accuracy.

Dense Embeddings

Lastly, the previous experiments are repeated with embeddings extracted from a fine-tuned DistilBERT model.

Method	a	Weighted Sampling	Avg. Accuracy (Dev.)	Avg. Accuracy
GM	/	No	0.92667(0.01165)	0.9476(0.0081)
GM	/	Yes	0.92833(0.01125)	0.9472(0.00559)
GM	0	No	0.935(0.00766)	**0.9484(0.0058)**
GM	0	Yes	**0.93583(0.01043)**	0.9464(0.0043)
Baseline	/	/	0.9225(0.01245)	0.946(0.00712)

Table 4.13.: Performance on Youtube (DistilBERT, CE Loss, $\tau = 0$)

Just as with the SMS data, the overall performance increases when using DistilBERT embeddings instead of TF-IDF encodings. All configurations of the GM algorithm included in Table 4.13 have similar test accuracies that are very close to the baseline performance. Potentially, this could mean that barely any samples are ignored or relabeled by the denoising method. However, this is apparently not the case. The following table summarizes the observed label dynamics for each setting.

a	Weighted Sampling	$\frac{I}{N}$	$\frac{CI}{FL}$	$\frac{FI}{CL}$	$\frac{CC}{CC+FC}$
/	No	0.27066(0.1506)	0.58888(0.0413)8	0.21845(0.17738)	/
/	Yes	0.26677(0.14463)	0.58132(0.003305)	0.21572(0.16721)	/
0	No	0.19049(0.03943)	0.56941(0.01467)	0.12955(0.04523)	0.05707(0.05552)
0	Yes	0.18375(0.05307)	0.57588(0.05285)	0.12061(0.05864)	0.08998(0.13788)

Table 4.14.: Statistics of Label Changes on Youtube (DistilBERT, CE Loss, $\tau = 0$)

The denoising method removes a considerable share of the encountered samples in each setting. For this dataset and encoding, the GM algorithm seems fairly capable of separating mislabeled samples from correctly annotated instances. However, the inclusion of an alternative label seems to be unsuccessful once again, since only very few of the conducted label changes are actually warranted. These observations align with the results obtained on the SMS dataset (cf. Table 4.8).

Lastly, the impact of the F_1-based losses on the performance will be analyzed.

Method	Update/Comp. Loss	a	Weighted Sampling	Avg. Accuracy (Dev.)	Avg. Accuracy
GM	F_1/CE	/	No	**0.945(0.00583)**	0.9496(0.0043)
GM	F_1/CE	/	Yes	0.94417(0.00562)	0.9508(0.00379)
GM	F_1/CE	0	No	0.94417(0.00562)	0.95(0.00471)
GM	F_1/CE	0	Yes	0.94417(0.00686)	0.95(0.00432)
GM	Macro-F_1/CE	/	No	0.92833(0.01125)	0.9524(0.0035)
GM	Macro-F_1/CE	/	Yes	0.93(0.01054)	0.952(0.00377)
GM	Macro-F_1/CE	0	No	0.92593(0.01329)	0.9524(0.00398)
GM	Macro-F_1/CE	0	Yes	0.9275(0.01115)	0.952(0.00377)
GM	F_1/F_1	/	No	0.94417(0.00562)	0.9516(0.0035)
GM	F_1/F_1	/	Yes	0.94417(0.00562)	0.9512(0.00368)
GM	Macro-F_1/Macro-F_1	/	No	0.93667(0.00583)	0.9516(0.00398)
GM	Macro-F_1/Macro-F_1	/	Yes	0.9375(0.00589)	0.9512(0.00368)
Baseline	F_1	/	/	0.94(0.008861)	**0.9548(0.00329)**
Baseline	Macro-F_1	/	/	0.9341/(0.0073)	0.9504(0.00572)
Baseline	CE	/	/	0.9225(0.01245)	0.946(0.00712)

Table 4.15.: Performance on Youtube (DistilBERT, F_1-Based Losses, $\tau = 0$)

Overall, the variation of the scores collected in Table 4.15 is fairly small. On the test set, a simple baseline trained with $\mathcal{L}_{F1_{pos.}}$ performed the best. Table 4.15 once again suggests, that choosing $\mathcal{L}_{F1_{pos.}}$ as the update loss for the GM algorithm is viable for dense representations of the data. This table concludes our experiments on single-label datasets.

4.4.3. Summary

A lot of results were presented in this chapter, therefore, this section aims to briefly summarize the main observations.
Dense representations generally resulted in better performances than sparse encodings. While the absolute performance gap to the baseline was larger when using TF-IDF features, as opposed to DistilBERT features, the denoising method was more successful in distinguishing between mislabeled and correctly labeled samples when dense representations were chosen.

The F_1-based losses can boost performance, both for the GM algorithm and the baseline. However, for the GM algorithm, updating with $\mathcal{L}_{F1_{pos.}}$ requires special care when sparse features are used. In particular, $\mathcal{L}_{F1_{pos.}}$ needs at least one positively labeled sample in the update batch in order to yield a non-zero batch gradient. This cannot be guaranteed, especially when the denoising method removes samples from the batch. One possible approach for dealing with this problem is tuning the removal threshold τ, as low values of τ will only remove samples with strongly negative gradient similarities from the batch.

This concludes the experiments on single-label datasets. The next chapter explores the multi-label setting.

5. Experiments on CheXpert

This section assesses the performance of the GM denoising algorithm on the CheXpert dataset which was introduced by Irvin et al. (2019) [20]. The CheXpert dataset is a multi-label medical imaging dataset provided by Stanford University[1] [20]. The CheXpert training set was annotated by a rule-based labeler, hence this dataset is weakly supervised and likely contains annotation errors [20]. Therefore, applying our proposed denoising strategy might boost performance on this dataset. The next section gives some more details about the CheXpert data collection based on the information provided in the original CheXpert paper [20] and the corresponding data sheet by Garbin et al. (2021) [77].

5.1. Data Description

The CheXpert dataset [20] is composed of chest radiographs from studies conducted at Stanford Hospital between October 2002 and July 2017. The images in the dataset are annotated with respect to 12 pathologies as well as the observations *Support Devices* and *No Finding* [77, 20]. Furthermore, the instances in the data collection are grouped by patient and by study. A study contains at least one image from the specific patient [77, 20].

The original CheXpert dataset was split into 3 parts: training, development and test [77, 20]. However, the test set is not publicly available since it is used as the evaluation set for the CheXpert competition [77, 20]. Therefore, following Giacomello et al. (2021) [78], the development set will be used as the test set in this thesis. Table 5.1 summarizes the number of observations of the original training and development set (cf. Garbin et al. 2021) [77, 20].

Split	Patients	Studies	Images
Train	64540	187641	223414
Dev.	200	200	234

Table 5.1.: Statistics of the Original CheXpert Split[2]

While the studies in the development and test set were hand-labeled by radiologists,

[1] The registration form for obtaining the data can be found at `https://stanfordmlgroup.github.io/competitions/chexpert/`, last accessed 2022-06-16

[2] Adapted from Garbin et al. 2021 [77], Table I

the annotations for the training set were generated by the *"CheXpert labeler"*[3], which was applied to the corresponding radiology reports [77, 20]. These reports are however not provided by Stanford University; only the chest radiographs are available [77, 20]. Since the corresponding report is based on the entire study, all of the chest radiographs contained in a particular study are annotated with the same labels [77, 20]. The CheXpert labeler performs three different steps to assign 14 labels to each study: *mention extraction*, *mention classification* and *mention aggregation* (cf. Irvin et al. 2019) [77, 20].

In the *mention extraction* phase, the labeler uses keyword lists, provided by radiologists, to find text segments mentioning one of the 14 observations in the report's Impression section, which is a short summary of its main findings [77, 20]. These extracted phrases are then matched against three other sets of rules in the *mention classification* stage in order to determine whether they express the presence or the absence of said observation, or potentially uncertainty about the observation [77, 20]. A mention can be classified as "uncertain" due to vagueness in the report or explicitly expressed uncertainty about the diagnosis by the radiologist [77, 20]. Irvin et al. (2019) [20] give the following examples for uncertainty:

> "heart size is stable" [20, p. 591]
>
> "diffuse reticular pattern may represent mild interstitial pulmonary edema" [20, p. 591]

In the last stage, a single label is assigned to each of the 14 observations [77, 20]. Irvin et al. (2019) [20] describe the phase as follows:

> "Observations with at least one mention that is positively classified in the report is assigned a positive (1) label. An observation is assigned an uncertain (u) label if it has no positively classified mentions and at least one uncertain mention, and a negative label if there is at least one negatively classified mention. We assign (*blank*) if there is no mention of an observation." [20, p. 592]

Note that in the raw data files, the uncertainty label is represented by -1 instead of "u" [77, 20]. To summarize, for training instances, four different labels can be assigned to each of the 14 observations, namely [77, 20]:

- 1: observation is present
- 0: observation is not present
- -1: it is uncertain whether the observation is present
- blank: the given observation was not mentioned in the Impression section

[3]The code for the labeler is available in this repository provided by Stanford University at `https://github.com/stanfordmlgroup/chexpert-labeler`, last accessed 2022-06-17

An exception is the "No Finding" class, which can only take a positive or a blank label [77, 20]. It is assigned label 1 if no pathology was labeled positive or uncertain and remains blank otherwise [77, 20]. Note that more pathologies than the 12 that correspond to dedicated classes in the dataset are taken into consideration for the "No Finding" label [77]. Therefore, it can happen that the "No Finding" label is blank despite there being no positive or uncertainty labels for a particular instance [77, 20]. The images in the development and test set were annotated by radiologists and contain no blank or uncertainty labels [77, 20]. There are no additional manual annotations for the CheXpert training set [77, 20], so the label changes made by the GM algorithm unfortunately cannot be compared to ground-truth labels.

In the context of this thesis, the inspected performance measures for this dataset were chosen to be macro-F_1, micro-F_1 and AUROC. Each performance measure gives different insights into the behaviour of the classifier [79, 70, 80]: The macro-F_1 score averages the individual F_1 scores of all labels. As a result, the performance on rare classes has a comparatively high impact on this evaluation metric, since the F_1 score of a seldomly appearing label contributes to the macro-F_1 just as much as the F_1 score of a frequently observed class [70]. The micro-F_1 measure on the other hand aggregates true positives, false negatives and false positives over all classes and computes a single F_1 score based on these accumulated values [70, 81, 82]. Therefore, rare classes have less impact on the micro-F_1 score than on the macro-F_1 value [70, 81, 82]. Note that for computing any type of F_1 measure, *decision thresholds* that indicate the minimum prediction value that is required to assign a positive label to an observation have to be defined [70, 81, 82]. On the contrary, the AUROC (Area Under the Receiver Operating Characteristics) score does not require setting a threshold [79]. The ROC curve of a binary classifier plots the false positive rate against the true positive rate achieved by the system for varying thresholds [79]. The AUROC metric measures the size of the area under the resulting curve [79]. AUROC values are defined on the interval $[0, 1]$, where 0.5 is the score achieved by random guessing [79, 20, 80]. Since the original CheXpert paper [20] computed class-wise AUROC scores, the macro-averaged AUROC scores will be reported to keep tables concise [80].

5.2. Data Preparation

The CheXpert dataset allows for several design choices regarding the data preprocessing. This section explains how the images and labels were prepared for the experiments in this thesis.

5.2.1. Label Preparation

For the following experiments, the uncertainty labels are mapped to label 1 and blank labels are assigned the value 0. For the blank labels, this reassignment is somewhat intuitive, since not every radiographic report will mention all 12 pathologies. If an observation is not even mentioned in the study report, there likely are no indications for

it in the corresponding images [77, 20]. Several approaches for handling the uncertainty labels were explored by Irvin et al. (2019) [20], including turning them to either 0 or 1, ignoring them, or keeping them as a separate class. Since the goal of the following experiments is not to achieve the best possible performance on CheXpert but to compare the multi-label version of the GM algorithm (i.e. Algorithm 2) to a baseline that is trained on the same labels, the rather simple option of assigning a positive label to all uncertain observations was chosen. Better overall performances could likely be achieved by using a more refined reassignment strategy [77, 20]. Moreover, only the labels for the 12 pathologies were used for training and evaluation.

5.2.2. Image Embeddings

For further simplification, the images in the dataset were considered independently, disregarding the structure provided by the studies. Since the original CheXpert development set will be used as the test set, observations of 7000 patients were split off from the training set to be used as a validation set. There are a total of 22815 chest radiographs in this new development set, which reduces the training set size to 200599 [77, 20]. The concrete label distribution for the training and validation data can be found below.

Pathology	Label 0 (Train)	Label 1 (Train)	Label 0 (Dev.)	Label 1 (Dev.)
Enlarged Cardiomediastinum	179690	20909	20523	2292
Cardiomegaly	168878	31721	19449	3366
Lung Opacity	100767	99832	11468	11347
Lung Lesion	191018	9581	21722	1093
Edema	142145	58454	16039	6776
Consolidation	162297	38302	18592	4223
Pneumonia	178254	22345	20351	2464
Atelectasis	140120	60479	16179	6636
Pneumothorax	180402	20197	20419	2396
Pleural Effusion	112905	87694	12694	10121
Pleural Other	195044	5555	22194	621
Fracture	191946	8653	21786	1029

Table 5.2.: Label Distribution in "New" Training and Development Set

The original CheXpert development set is relatively small with only 234 examples [77, 20]. As a result, for *Lung Lesion* and *Pleural Other*, there is only one positive observation, while *Fracture* was never marked as positive [77, 20]. These three classes are therefore excluded when reporting the performance metrics, since especially the macro-F_1 score could be greatly affected by them [70]. The concrete label distribution in our test set can be found in Table 5.3.

Pathology	Label 0 (Test)	Label 1 (Test)
Enlarged Cardiomediastinum	125	109
Cardiomegaly	166	68
Lung Opacity	108	126
Lung Lesion	233	1
Edema	189	45
Consolidation	201	33
Pneumonia	226	8
Atelectasis	154	80
Pneumothorax	226	8
Pleural Effusion	167	67
Pleural Other	233	1
Fracture	234	0

Table 5.3.: Label Distribution in "New" Test Set (i.e. the Original Development Set)[4]

In order to use the CheXpert images with a simple model architecture, the chest radiographs first need to be encoded [78]. Giacomello et al. (2021) [78] found that using fine-tuned convolutional neural networks (CNNs) as feature extractors and training a separate classifier with the obtained *image embeddings* can result in a good performance on the CheXpert data. Therefore, this approach was adopted in this thesis' experiments.

Image embeddings are low-dimensional representations that map each image to a single vector [78]. Such encodings can be created by removing the classification layer of a particular CNN and retrieving the output of this modified architecture for each image, as was done in the experiments by Giacomello et al. (2021) [78]. There is a large selection of CNN architectures that could be used to extract these encodings. For the experiments in this thesis, a pre-trained EfficientNet-B0 [83] from the torchvision library was chosen[5].

The EfficientNet models, proposed by Tan et al. (2019) [83], achieve comparable performance to other families of CNNs, such as ResNets [84] or DenseNets [85], while using much fewer parameters. Tan et al. (2019) [83] accomplished this by developing a new strategy for adjusting the size of a CNN depending on how many resources are available. To improve the performance of a particular CNN architecture, usually, either the number of layers, the width of the network or the resolution of the input images is increased [83]. Tan et al. (2019) [83] suggested adjusting all three of these aspects simultaneously, instead of only one, by using their *compound scaling method*. Their smallest network, EfficientNet-B0, was developed by balancing performance with the number of floating point operations. The remaining EfficientNet models, B1 to B7, were constructed from this baseline by using compound scaling [83]. On ImageNet, EfficientNet-B1 showed a 1% top-1 accuracy improvement over ResNet-152 despite using only 7.8 million parameters as opposed to 60 million. Similar observations were made for the other variants of EfficientNet, indicating their effectiveness for learning with low computational resources

[4] Adapted from Garbin et al. 2021 [77], Table V

[5] `https://pytorch.org/vision/0.8/models.html`, last accessed 2022-06-20

[83].

The torchvision implementation of EfficientNet-B0 used for the experiments in this thesis was pre-trained on ImageNet. While the images contained in ImageNet greatly differ from chest radiographs, the pre-training still gives good intialization as was shown by Ke et al. (2021) [86]. EfficientNet-B0 requires input images of size 224x224 [83]. Moreover, in order to use the pre-trained version provided by torchvision, the images should be normalized with the mean and standard deviation from ImageNet [87]. Therefore, appropriate transformations were applied to all chest radiographs in the CheXpert dataset. The EfficientNet was fine-tuned on the training data for two epochs, using Adam optimizer with learning rate 0.0001 and batch size 16, which are the hyperparameters used in the original CheXpert paper [20]. Once the fine-tuning was completed, embeddings were extracted from the penultimate layer of the network. The classification layer of EfficientNet-B0 takes 1280 inputs, therefore, the resulting image embeddings are 1280-dimensional vectors [83].

5.3. Experimental Setup

Just like in the single-label case, only the classification head of the CNN is re-trained with the proposed denoising method, while the remaining parameters are frozen after the EfficientNet was fine-tuned for two epochs. The number of input nodes for the classification layer is equal to the embedding size, namely 1280, and each of the 12 pathologies is represented by an output node. The 12 model outputs for a sample t can therefore be written as indicated below [62]. The same notation as in the single-label case is used.

$$
\begin{aligned}
z_{t,1} &= \sum_{f=1}^{F} (w_{f,1} x_{t,f}) + b_1 \\
z_{t,2} &= \sum_{f=1}^{F} (w_{f,2} x_{t,f}) + b_2 \\
&\vdots \\
z_{t,12} &= \sum_{f=1}^{F} (w_{f,12} x_{t,f}) + b_{12}
\end{aligned}
$$

Here, $F = 1280$. Since the CheXpert data is significantly larger than both SMS and Youtube [20, 7], the number of runs per setting is reduced to 3. Moreover, the batch size will be fixed to 128 and the learning rate is set to 0.001 for all experiments. Each model is trained for 5 epochs. The best state of the model, measured by the validation loss, is loaded after these 5 epochs and evaluated on the test set. Note that this is a slightly different procedure than in the single-label setting, where the actual performance metric, namely F_1 or accuracy, was used to determine the best epoch. This change was made since now three different metrics are analyzed.

5.3.1. Examined Loss Functions

Akin to the experiments on single-label datasets, different loss functions are examined, namely binary cross-entropy (BCE) as well as a soft macro-F_1 loss and a soft micro-F_1 loss, based on Bénédict et al. (2021) [19]. The main difference to the single-label losses is that now a sigmoid activation is applied to the output, instead of a softmax, since multiple classes can be positive for a single instance [19].

BCE Loss

The BCE loss computed on batch $\mathcal{B}$, with batch size M and K labels per instance, takes the form[6]

$$\mathcal{L}^*_{BCE} = \frac{1}{MK} \sum_{t=1}^{M} \sum_{k=1}^{K} l^*_{t,k}$$

where

$$l^*_{t,k} = -[\log(\sigma(z_{t,k}))y_{t,k} + \log(1 - \sigma(z_{t,k}))(1 - y_{t,k})]$$

and $\sigma(z_{t,k}) = \frac{1}{1+e^{-z_{t,k}}}$ [88, 89].

Note that this loss is decomposable with respect to the individual samples and labels [89]. The loss follows a so-called *"one-vs-all"* approach, which simplifies the multi-label problem to several binary classification tasks, ignoring dependencies between the classes [19, 89].

The proposed denoising method, as described in Algorithm 2, should allow to ignore individual labels $y_{t,k}$ for the update. If, for example, the last label of sample t, $y_{t,K}$, should be ignored, all components of the sample gradient corresponding to the weights and biases connected to label K need to be zeroed out. Therefore, the matrices representing the instance-specific gradients of sample t with respect to the learnable parameters should look like the following expressions, given the network architecture used in the experiments (cf. A.1.2) [90, 89].

$$\begin{bmatrix} (\sigma(z_{t,1}) - 1)x_{t,1}y_{t,1} + \sigma(z_{t,1})x_{t,1}(1 - y_{t,1}) & \dots & (\sigma(z_{t,1}) - 1)x_{t,F}y_{t,1} + \sigma(z_{t,1})x_{t,F}(1 - y_{t,1}) \\ & \vdots & \\ (\sigma(z_{t,K-1}) - 1)x_{t,1}y_{t,K-1} + \sigma(z_{t,K-1})x_{t,1}(1 - y_{t,K-1}) & \dots & (\sigma(z_{t,K-1}) - 1)x_{t,F}y_{t,K-1} + \sigma(z_{t,K-1})x_{t,F}(1 - y_{t,K-1}) \\ 0 & \dots & 0 \end{bmatrix}$$

for weights and

$$\begin{bmatrix} (\sigma(z_{t,1}) - 1)y_{t,1} + \sigma(z_{t,1})(1 - y_{t,1}) \\ \vdots \\ (\sigma(z_{t,K-1}) - 1)y_{t,K-1} + \sigma(z_{t,K-1})(1 - y_{t,K-1}) \\ 0 \end{bmatrix}$$

[6] Formula from `https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html`, last accessed 2022-06-24

for biases. In order to achieve this, a masked version of $\mathcal{L}^*_{BCE}$, which can be used as the update loss for the GM algorithm, is defined in the following.

Let $\widetilde{M}_k$ be the number of samples in the batch, for which the label for class k was not ignored. Moreover, let $\widetilde{y}_{t,k}$ denote the value of label k of sample t after the label reassignment. Since in the multi-label setting, no alternative label is used, $\widetilde{y}_{t,k}$ is either the original label, $y_{t,k}$, or the label i, which signals that this particular label should be ignored for the update. The masked BCE loss used in the experiments can be written as[7]

$$\overline{\mathcal{L}^*}_{BCE}(\mathcal{B}) = \frac{1}{K}\sum_{k=1}^{K}\frac{1}{\widetilde{M}_k}\sum_{t=1}^{M} -[\log(\sigma(z_{t,k}))y_{t,k} + \log(1-\sigma(z_{t,k}))(1-y_{t,k})]\mathbb{1}_{(\widetilde{y}_{t,k}\neq i)}$$

Using this formulation, if $\widetilde{y}_{t,k} = i$, sample t does not contribute to the loss associated with label k. Moreover, the sample-specific gradients for weights and biases connected to label k will be 0 for sample t.

Soft Macro-F_1 Loss

Another examined loss function is a soft macro-F_1 loss[8], as described by Bénédict et al. (2021) [19]. It has the theoretical advantage over the BCE loss that it directly approximates a multi-label performance measure [19]. The loss can be defined as follows (cf. Bénédict et al. 2020) [19].

$$\mathcal{L}^*_{F_{1_{macro}}}(\mathcal{B}) = 1 - \frac{1}{K}\sum_{k=1}^{K}\frac{2\widetilde{tp}_k}{2\widetilde{tp}_k + \widetilde{fn}_k + \widetilde{fp}_k + \epsilon}$$

where

$$\widetilde{tp}_k = \sum_{t=1}^{M} y_{t,k}\sigma(z_{t,k})$$
$$\widetilde{fp}_k = \sum_{t=1}^{M}(1-y_{t,k})\sigma(z_{t,k})$$
$$\widetilde{fn}_k = \sum_{t=1}^{M} y_{t,k}(1-\sigma(z_{t,k}))$$

This formulation is very similar to $\mathcal{L}_{F_{1_{macro}}}$, as defined in Section 4.3.3, but here a sigmoid activation is used as opposed to the softmax.

If this soft macro-F_1 loss should be used for the model update, a masked version is again needed. The masked loss can be defined by adapting the estimated confusion matrix values to not take ignored labels into account and subsequently plugging them into the

[7] Formulation inspired by Irvin et al. (2019) [20]

[8] An implementation of this loss can be found at `https://github.com/ashrefm/multi-label-soft-f1/blob/master/Multi-Label%20Image%20Classification%20in%20TensorFlow%202.0.ipynb`, last accessed 2022-06-21

formula for $\mathcal{L}^*_{F_{1_{macro}}}$. For the experiments in this theses, the masked confusion matrix estimates were computed as indicated below.

$$\overline{tp}_k = \sum_{t=1}^{M} y_{t,k}\sigma(z_{t,k})\mathbb{1}_{(\widetilde{y}_{t,k}\neq i)}$$
$$\overline{fp}_k = \sum_{t=1}^{M} (1-y_{t,k})\sigma(z_{t,k})\mathbb{1}_{(\widetilde{y}_{t,k}\neq i)}$$
$$\overline{fn}_k = \sum_{t=1}^{M} y_{t,k}(1-\sigma(z_{t,k}))\mathbb{1}_{(\widetilde{y}_{t,k}\neq i)}$$

Soft Micro-F_1 Loss

While Bénédict et al. (2021) [19] describe a label-wise soft F_1 loss, it should be possible to formulate a micro-averaged version by aggregating the estimated confusion matrix entries over all labels as indicated below.

$$\widetilde{tp} = \sum_{t=1}^{M}\sum_{k=1}^{K} y_{t,k}\sigma(z_{t,k})$$
$$\widetilde{fp} = \sum_{t=1}^{M}\sum_{k=1}^{K} (1-y_{t,k})\sigma(z_{t,k})$$
$$\widetilde{fn} = \sum_{t=1}^{M}\sum_{k=1}^{K} y_{t,k}(1-\sigma(z_{t,k}))$$

$$\mathcal{L}^*_{F_{1_{micro}}}(\mathcal{B}) = 1 - \frac{2\widetilde{tp}}{2\widetilde{tp} + \widetilde{fn} + \widetilde{fp} + \epsilon}$$

A masked version of the micro-F_1 loss can be constructed analogously to the macro-F_1. The reduced estimated confusion matrix entries in this case are given by the following expressions.

$$\overline{tp} = \sum_{t=1}^{M}\sum_{k=1}^{K} y_{t,k}\sigma(z_{t,k})\mathbb{1}_{(\widetilde{y}_{t,k}\neq i)}$$
$$\overline{fp} = \sum_{t=1}^{M}\sum_{k=1}^{K} (1-y_{t,k})\sigma(z_{t,k})\mathbb{1}_{(\widetilde{y}_{t,k}\neq i)}$$
$$\overline{fn} = \sum_{t=1}^{M}\sum_{k=1}^{K} y_{t,k}(1-\sigma(z_{t,k}))\mathbb{1}_{(\widetilde{y}_{t,k}\neq i)}$$

5.3.2. Tuning of Decision Thresholds

In the multi-label setting, multiple classes can be assigned a positive label per instance [49]. Therefore, each class requires a threshold, which dictates the predicted probability that needs to be exceeded for the corresponding label to be marked as positive [19, 70, 81, 82]. This threshold can either be constant across all classes or class-specific [19]. In the following experiments, the threshold for each of the 12 pathologies is tuned on the development set.

For optimizing the macro-F_1 score, the threshold of each class can be tuned individually, since the F_1 scores for each class are simply averaged in this case [70, 81, 82]. Therefore, the macro-F_1 score is optimized when all class-wise F_1 scores are maximized [70, 81, 82]. In the experiments, we evaluate decision thresholds ranging from 0 to 1 with a step size of 0.04. For each class, the threshold that gives the best F_1 score on the development set is chosen. The resulting thresholds will be used for the evaluation of the macro-F_1 score on the test set.

Tuning the thresholds to maximize the micro-F_1 score is more complicated, since it cannot be decomposed into class-wise contributions [91, 70, 81, 82]. Therefore, the "best" threshold value for one class depends on the current thresholds of the other labels [91, 70, 81, 82]. While some papers simply recycle the optimized thresholds with respect to the macro-F_1 for evaluating the micro-F_1 [81], Pillai et al. (2012, 2013) [81, 82] developed an algorithm for selecting class-specific thresholds on the development set that specifically maximize the micro-F_1 score. Their procedure goes as follows[9] [81, 82]: In each iteration, the algorithm selects the best threshold for each class, with respect to the micro-F_1 score, from a pre-defined set of thresholds while keeping the remaining thresholds fixed. This process is repeated until the chosen thresholds do not change anymore [81, 82]. Algorithm 4 gives the concrete formulation after Pillai et al. (2012, 2013) [81, 82]. In the algorithm below, $F_{1_{micro}}(\hat{y}_1, ..., \hat{y}_{|\mathcal{D}_V|}; t_1, ..., t_K)$ denotes the micro-F_1 score with respect to the predicted probabilities $\hat{y}_1, ..., \hat{y}_{|\mathcal{D}_V|}$ on the validation set $\mathcal{D}_V$ and the class-wise thresholds $t_1, ..., t_K$. Note that in the multi-label setting, $\hat{y}_1, ..., \hat{y}_{|\mathcal{D}_V|}$ are K-dimensional vectors.

[9] The original formulation of the algorithm can be found as Algorithm 1 in "F-Measure Optimisation in Multi-Label Classifiers" (Pillai et al. (2012)) [81]

Input : prediction scores $\hat{y}_1, ..., \hat{y}_{|\mathcal{D}_V|}$,
set of possible threshold values
Output : optimized per-class thresholds
Initialize $t_1, ..., t_K$ as the smallest possible threshold values;
while *True* **do**
  $changes \leftarrow 0$;
  for *k=1,...,K* **do**
    if $t_k \neq \underset{T \geq t_k}{\text{argmax}}\, F_{1_{micro}}(\hat{y}_1, ..., \hat{y}_{|\mathcal{D}_V|}; t_1, ...t_{k-1}, T, t_{k+1}, ...t_K)$ **then**
      $t_k \leftarrow \underset{T \geq t_k}{\text{argmax}}\, F_{1_{micro}}(\hat{y}_1, ..., \hat{y}_{|\mathcal{D}_V|}; t_1, ...t_{k-1}, T, t_{k+1}, ...t_K)$;
      $changes \leftarrow changes + 1$
    end
  end
  if $changes = 0$ **then**
    end procedure
  end
end
return *optimized thresholds* $t_1, ..., t_K$

Algorithm 4: Tuning of the Prediction Thresholds for Micro-F_1 Score after Pillai et al. (2012, 2013)

In the experiments, the set of possible thresholds defined for the optimization of the micro-F_1 score is the same as for the macro-F_1 score, namely $\{0, 0.04, 0.08, ..., 0.96, 1\}$.

Note that, according to Bénédict et al. (2021) [19], updating with the soft macro-F_1 loss naturally pushes the predicted probabilities towards 0 and 1. As a result, a simple 0.5 threshold across all classes should be sufficient to achieve good F_1 scores[10] [19, 74]. For the models trained with either the soft macro- or micro-F_1 loss, the results will therefore be reported for both the tuned thresholds and a constant 0.5 threshold across classes. For the settings that update with the BCE loss, only the performance achieved by using the tuned thresholds will be reported.

Now that the overall experimental setup was clarified, the following section presents the results achieved on the CheXpert data. Due to space restrictions, only the performances on the test set are reported. The corresponding scores on the development set can be found in the Appendix (cf. A.2).

[10] cf. `https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d`, last accessed 2022-06-30

5.4. Results

5.4.1. BCE Loss

Table 5.4 summarizes the results obtained on the CheXpert dataset by solely using the BCE loss for the baseline models and the GM algorithm. The default removal threshold, $\tau = 0$, was used for creating the table. As mentioned before, the table reports average scores acquired over three runs with different seeds. Like in the single-label case, the standard deviations of the scores are given in parentheses. All reported values were computed on the test set. Please consult the appendix for the corresponding development performances.

Method	Update/Comp. Loss	Threshold Tuning	Avg. Macro-F_1	Avg. Micro-F_1	Avg. AUROC
GM	BCE/BCE	Yes	0.54932(0.00608)	0.62326(0.00166)	0.821(0.00708)
Baseline	BCE	Yes	**0.55214(0.00613)**	**0.62481(0.00327)**	**0.82435(0.00456)**

Table 5.4.: Test Performance on CheXpert (BCE Loss, $\tau = 0$)

Judging from Table 5.4, there does not seem to be a large performance difference between models trained with and without gradient matching. The baseline slightly outperforms the models trained with the GM algorithm with respect to all three performance measures.

Taking a look at the per-class test AUROC scores might give insights on how denoising with the GM algorithm affects the performance on the individual classes. Therefore, the individual AUROC scores per class are provided below.

Pathology	Avg. AUROC (Baseline)	Avg. AUROC (GM)
Enlarged Cardiomediastinum	**0.47251(0.02828)**	0.46642(0.02285)
Cardiomegaly	**0.80558(0.01090)**	0.80079(0.00674)
Lung Opacity	0.91155(0.00427)	**0.91165(0.00338)**
Edema	**0.92914(0.00177)**	0.92702(0.00124)
Consolidation	**0.89879(0.00645)**	0.88894(0.00289)
Pneumonia	0.81158(0.01986)	**0.82098(0.05164)**
Atelectasis	**0.81859(0.00534)**	0.80384(0.01044)
Pneumothorax	**0.83684(0.01435)**	0.83499(0.00470)
Pleural Effusion	**0.93455(0.00537)**	0.93437(0.00069)

Table 5.5.: Test AUROC per Class (BCE Loss, $\tau = 0$)

All classes, except for *Enlarged Cardiomediastinum*, have decent AUROC values. Many papers unfortunately only report the individual AUROC scores on the five classes evaluated in the CheXpert competition, namely *Atelectasis*, *Cardiomegaly*, *Consolidation*, *Edema* and *Pleural Effusion* [77, 20]. Wei and Jethani (2019) [92], however, present their AUROC scores for all classes. They also achieved worse performance on *Enlarged Cardiomediastinum* compared to the other pathologies, which matches our observations.

For most pathologies, the average AUROC value achieved by baseline models is marginally better than the scores attained by applying our denoising strategy. The only class

that noticeably benefited from the denoising method is *Pneumonia*, but the improvement over the baseline is very small and could just be due to chance or the low number of runs.

Unfortunately, it is not possible to check how the label changes relate to the gold labels on the CheXpert data. However, taking a look at the label dynamics in an exemplary run of the algorithm can nevertheless be insightful. Figure 5.1 therefore provides an overview of the conducted label changes in a single run of the GM algorithm. Akin to the plots provided in Chapter 2, the depicted values sum up to one for a given batch. The displayed statistics were smoothed over 10 batches in order to facilitate the interpretation of the figure.

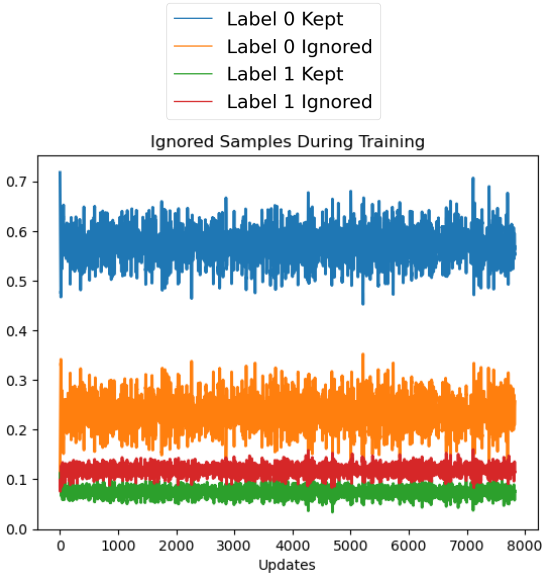


Figure 5.1.: Label Changes during Training for CheXpert (BCE/BCE, $\tau = 0$)

Interestingly, the label dynamics stay fairly constant throughout the training process. Moreover, negative labels seem to have a higher change of being kept than the less frequent positive labels. Assuming that the denoising method is indeed capable of finding labeling errors, this seems sensible: Since all uncertainty labels were mapped to the positive label in the data preparation phase, the resulting number of positive labels is likely inflated compared to the ground-truth.

As mentioned before, for all results collected in Table 5.4, $\tau = 0$ was used. Adjusting τ could potentially slightly improve the performance of the GM algorithm. Figure 5.2 therefore shows how the performance measures and label changes are affected by the value of τ. Due to the increased data size compared to SMS and Youtube [7], this plot only depicts a single run of the denoising algorithm per threshold. The learning rate was kept fixed at 0.001 for the creation of this figure and the batch size was 128.

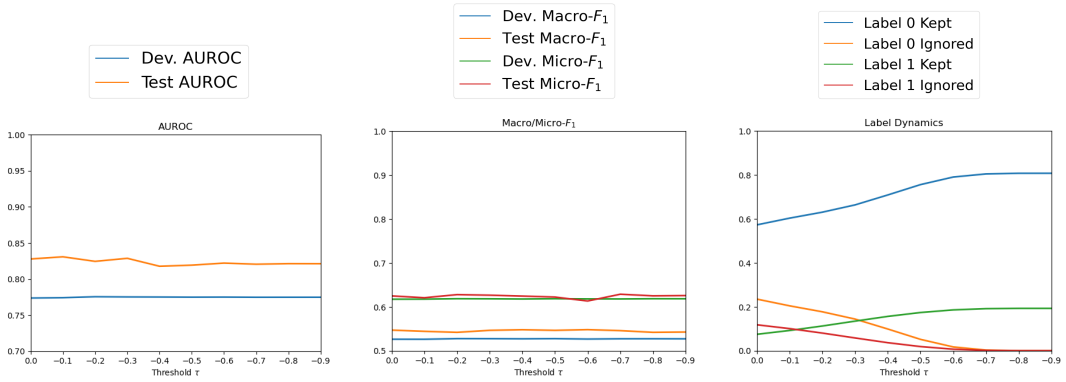


Figure 5.2.: Effect of Removal Threshold Value for CheXpert (BCE/BCE)

None of the performance metrics seem to be strongly affected by the value of τ. This observation matches the fact that the results were already very close to the baseline for $\tau = 0$, which is the value that tends to remove the most samples. Since the threshold tuning did not boost the performance of the GM algorithm, it seems that our denoising method is not useful for this dataset when solely the BCE loss is used.

The next section explores whether changing the loss function, for either the model update or the gradient comparison, results in a more noticeable performance difference between the GM algorithm and the baseline.

5.4.2. F_1-Based Losses

Macro

The following table reports the results obtained by using the soft macro-F_1 loss. For models that update with this loss, the performances are reported for tuned thresholds and a simple 0.5 threshold across all classes. The average scores for the baseline models fit with cross-entropy are again included in this table as a reference.

Method	Update/Comp. Loss	Threshold Tuning	Avg. Macro-F_1	Avg. Micro-F_1	Avg. AUROC
GM	Macro-F_1/Macro-F_1	No	0.53751(0.00456)	0.59423(0.00555)	0.76981(0.00749)
GM	Macro-F_1/Macro-F_1	Yes	0.54366(0.00571)	0.58187(0.00342)	0.76981(0.00749)
GM	BCE/Macro-F_1	Yes	**0.56447(0.00336)**	0.62866(0.00303)	**0.83652(0.01103)**
Baseline	Macro-F_1	No	0.55879(0.00075)	**0.63613(0.00029)**	0.82446(0.00089)
Baseline	Macro-F_1	Yes	0.55508(0.00395)	0.61586(0.00477)	0.82446(0.00089)
Baseline	BCE	Yes	0.55214(0.00613)	0.62481(0.00327)	0.82435(0.00456)

Table 5.6.: Test Performance on CheXpert (Macro-F_1 Loss, $\tau = 0$)

Looking at the baseline scores, using the soft macro-F_1 loss results in very similar performances compared to the cross-entropy loss, even when the thresholds are not tuned. This observation indicates that using the soft macro-F_1 loss to update the baseline models

can potentially be a viable alternative to threshold tuning, which could prove to be very time-efficient on data collections with even larger development sets[11] [19, 74]. Note that tuning the thresholds does not necessarily improve the test performance, since the optimization happens on the development set [81].

For the models trained with the GM algorithm, all scores noticeably decrease when the macro-F_1 loss is used to update. This reflects some results found in the single-label case, suggesting that F_1-based losses may not be a good choice for the update loss when the GM algorithm is applied. Keeping the BCE loss as the update loss and merely using the macro-F_1-based loss to conduct the gradient comparison yields much better test scores. In fact, this setting gives the best performance in terms of macro-F_1 and AUROC seen in the experiments so far. This result demonstrates that even unsuitable update losses for the GM algorithm can be leveraged by using them as the comparison loss.

The individual test AUROC values per class are presented in Table 5.7 to give some further insights into the poor performance of $\overline{\mathcal{L}^*}_{F_{1_{macro}}}$ as an update loss for the GM algorithm.

Pathology	Avg. AUROC (Baseline)	Avg. AUROC (GM, Macro-F_1/MacroF_1)	Avg. AUROC (GM, BCE/Macro-F_1)
Enlarged Cardiomediastinum	0.48132(0.00331)	0.4781(0.034842)	**0.5062(0.048)**
Cardiomegaly	0.79595(0.00174)	0.79613(0.01276)	**0.81423(0.00416)**
Lung Opacity	0.90888(0.00095)	0.3833(0.00869)	**0.92203(0.00087)**
Edema	0.93059(0.00127)	0.93208(0.00162)	**0.93255(0.00034)**
Consolidation	0.8804(0.00308)	0.88944(0.00139)	**0.90507(0.01108)**
Pneumonia	0.82172(0.00526)	**0.85371(0.02644)**	0.83776(0.03523)
Atelectasis	0.81307(0.00575)	0.82275(0.00316)	**0.8342(0.00241)**
Pneumothorax	**0.8549(0.00169)**	0.842(0.01559)	0.84329(0.01439)
Pleural Effusion	0.93334(0.00053)	0.9308(0.00048)	**0.93339(0.00083)**

Table 5.7.: Test AUROC per Class (Macro-F_1 Loss, $\tau = 0$)

Interestingly, Table 5.7 reveals that the subpar averaged scores achieved by updating with $\overline{\mathcal{L}^*}_{F_{1_{macro}}}$ seem to primarily hinge on one class, *Lung Opacity*. It turns out that the predicted probability for this class is very close 1 for all test samples, which explains the low AUROC score. It will be interesting to see how well models trained with $\overline{\mathcal{L}^*}_{F_{1_{micro}}}$ perform on this particular class in comparison.

[11] Note that this result was also observed in this blog post `https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d`, last accessed 2022-06-30

Micro

Lastly, the results obtained by using the soft micro-F_1 loss are reported.

Method	Update/Comp. Loss	Threshold Tuning	Avg. Macro-F_1	Avg. Micro-F_1	Avg. AUROC
GM	Micro-F_1/Micro-F_1	No	0.45795(0.0063)	0.58933(0.00481)	0.76094(0.00405)
GM	Micro-F_1/Micro-F_1	Yes	0.55018(0.00298)	0.58933(0.00176)	0.76094(0.00405)
GM	BCE/Micro-F_1	Yes	0.56445(0.01992)	**0.63121(0.00332)**	0.83582(0.00541)
Baseline	Micro-F_1	No	0.51225(0.01492)	0.62351(0.0051)	**0.8412(0.00565)**
Baseline	Micro-F_1	Yes	**0.59003(0.00499)**	0.62129(0.00214)	**0.8412(0.00565)**
Baseline	BCE	Yes	0.55214(0.00613)	0.62481(0.00327)	0.82435(0.00456)

Table 5.8.: Test Performance on CheXpert (Micro-F_1 Loss, $\tau = 0$)

Interestingly, training the baseline models with this loss achieves the best average macro-F_1 and AUROC score seen so far, but does not improve the value of the micro-F_1 compared to the BCE loss. This seems counterintuitive, but there are a lot of factors that could play into this result, like the small size of the test set or the relatively coarse threshold tuning. Moreover, the thresholds are tuned on the noisy development set, so the chosen threshold values might not generalize well to the test set. For the models trained with gradient matching, the BCE loss again seems to be the preferable update loss. The best micro-F_1 recorded in Table 5.8 was achieved by making use of gradient matching and updating the model with the BCE loss, while computing the compared gradients with respect to $\mathcal{L}^*{}_{F_{1_{micro}}}$.

The individual AUROC values are given below.

Pathology	Avg. AUROC (Baseline)	Avg. AUROC (GM, Micro-F_1/Micro-F_1)	Avg. AUROC (GM, BCE/Micro-F_1)
Enlarged Cardiomediastinum	**0.62613(0.03188)**	0.58982(0.01061)	0.50092(0.0639)
Cardiomegaly	0.78792(0.00583)	0.79066(0.00882)	**0.81302(0.00553)**
Lung Opacity	0.90631(0.00197)	0.39627(0.00966)	**0.92308(0.00037)**
Edema	0.9293(0.00073)	0.93135(0.00038)	**0.93416(0.00071)**
Consolidation	0.88427(0.00738)	0.87266(0.00571)	**0.89015(0.01317)**
Pneumonia	0.83555(0.01302)	0.69358(0.00561)	**0.85490(0.0366)**
Atelectasis	0.81489(0.00169)	0.80835(0.00427)	**0.82944(0.00719)**
Pneumothorax	**0.8549(0.00787)**	0.83499(0.01391)	0.84255(0.00546)
Pleural Effusion	0.93155(0.00191)	0.9308(0.00125)	**0.93419(0.00121)**

Table 5.9.: Test AUROC per Class (Micro-F_1 Loss, $\tau = 0$)

Updating with the micro-F_1 loss noticeably boosted the performance on *Enlarged Cardiomediastinum*, which is the hardest class to predict, compared to the other loss functions examined so far. This effect is particularly noticeable for the baseline. On the other hand, the class *Lung Opacity* suffers from updating with the soft micro-F_1 loss when gradient matching is applied, which reflects the results achieved with the macro-averaged version of the loss. A look at the label changes performed during training reveals that negative labels are mostly ignored for this class, which possibly explains why the predictions for this class are so heavily pushed towards 1. Why this class in particular suffers from this phenomenon is however not entirely clear, since the class has a fairly even distribution of positive and negative labels in both the training and the test set.

To wrap up the experiments, Table 5.10 compares the label dynamics for all explored settings of the gradient matching algorithm. Note that the values in each row sum up to one as they were computed equivalently to the lines depicted in Figure 5.1.

Update/Comp. Loss	Label 0 Ignored	Label 1 Ignored	Label 0 Kept	Label 1 Kept
BCE/BCE	0.23394(0.00043)	0.117799(0.00022)	0.57342(0.00043)	0.07485(0.00022)
Macro-F_1/Macro-F_1	0.21076(0.00228)	0.1115(0.00415)	0.5966(0.00228)	0.08114(0.00415)
BCE/Macro-F_1	0.39843(0.00132)	0.05407(0.00034)	0.40893(0.00132)	0.13857(0.00034)
Micro-F_1/Micro-F_1	0.26526(0.00102)	0.0893(0.00198)	0.5421(0.00102)	0.10344(0.00198)
BCE/Micro-F_1	0.21507(0.00038)	0.07002(0.00014)	0.59229(0.00038)	0.12262(0.00014)

Table 5.10.: Label Dynamics on CheXpert (All Loss Combinations, $\tau = 0$)

Judging from Table 5.10, using either the BCE loss or the Macro-F_1 loss for both the update and the gradient comparison results in more positive labels being ignored than in the other settings. Exploring whether this effect is specific to this dataset or whether a theoretically founded explanation for this phenomenon can be found could be an interesting avenue for future work.

5.4.3. Summary

All in all, no strong conclusions can be drawn about the effectiveness of the GM algorithm in the multi-label setting from the results presented in this chapter. However, the experiments affirmed that F_1-based loss functions tend to be suboptimal update losses when the GM algorithm is applied, at least at the default removal threshold $\tau = 0$. Moreover, the obtained results show that it can be beneficial to choose different loss functions for the update and the computation of the compared gradients. For the baseline models, updating with the F_1-based losses generally worked well, which mirrors the results found in the single-label case.

This concludes the experiments conducted in this thesis. The next chapter aims to summarize the main insights gained throughout the thesis and outlines some possibilities for future work.

6. Conclusion

The results obtained in Chapters 4 and 5 give some interesting insights into denoising with gradient matching but also leave some unanswered questions. The next section aims to shortly summarize the main takeaways from the conducted experiments.

6.1. Main Observations

On single-label datasets, we observed fairly large improvements in performance over the baseline when sparse TF-IDF features were used and the model was updated with the regular cross-entropy loss. In particular, weighted comparison batch sampling proved useful in this scenario, both on the imbalanced SMS dataset and the Youtube dataset, which has a more even class distribution. Despite the performance gain over the baseline, a large fraction of the correctly labeled samples seen during training was ignored on both datasets. When DistilBERT embeddings were used instead of TF-IDF representations, the GM algorithm was generally more successful in distinguishing between correctly labeled and mislabeled samples. However, for dense features, the improvements in performance over the baseline scores were generally small, which is surprising considering that the algorithm cleaned the data more effectively than in the TF-IDF setting.

The experiments on the single-label datasets further revealed that updating with $\mathcal{L}_{F1_{pos.}}$, which aims to directly optimize the F_1 score of the positive class, can generally boost performance but needs to be handled with care when the GM algorithm is applied to sparse encodings. While the default removal threshold $\tau = 0$ worked well for most settings investigated throughout the thesis, using this threshold value on the SMS dataset resulted in very poor scores when the update was conducted with $\mathcal{L}_{F1_{pos.}}$ on sparse features (cf. Table 4.6, Figure 4.2). Tuning the removal threshold was crucial in this particular scenario. Intuitively, it makes sense that F_1-based update losses can be problematic when the GM algorithm is applied, since they cannot be decomposed into sample-wise contributions and strongly depend on the label distribution in the update batch [19]. Changing the annotations with gradient matching can drastically alter the label distribution in the update batch and, in extreme cases, lead to scenarios where the model is not properly updated. Nevertheless, $\mathcal{L}_{F1_{pos.}}$ worked well as an update loss for the GM algorithm when the texts were represented by dense DistilBERT features.

The experiments on the CheXpert dataset unfortunately did not reveal much about the effectiveness of our method in the multi-label setting. However, the results affirmed that the F_1-based losses can be suboptimal update losses for the GM algorithm, at least at the default removal threshold. Nevertheless, the soft F_1 losses proved to be suitable comparison losses for the GM algorithm. Moreover, the baseline results on CheXpert

confirmed that models trained with the soft macro-F_1 loss, as suggested by Bénédict et al. (2021) [19], perform well at a simple 0.5 decision threshold across all classes, which potentially eliminates the need for decision threshold tuning.

To summarize, in most cases, applying the GM algorithm lead to test scores which were similar to or slightly better than the baseline performance. However, for sparse features, the algorithm did not effectively distinguish between mislabeled and correctly labeled samples, even when comparatively large performance gaps were achieved.

6.2. Potential Relation to Other Topics

The observations summarized above indicate that our proposed approach is not particularly successful in its goal of removing or correcting mislabeled samples. However, the conducted experiments show that the method could still be a useful addition to the training process. After all, at least a small improvement in performance was achieved compared to the baseline models in most scenarios. Our experiments showed that removing large numbers of correctly labeled samples from the batches generally did not have a negative impact on the model performance. The GM algorithm could therefore potentially be interpreted as a method that constructs effective update batches by removing samples that have unusual gradients, mislabeled or not. The observed performance gains might be due to the method only keeping the samples in a given batch that are currently beneficial for the learning process.

Potentially, the method could also be linked to outlier detection. Gradient-based methods have already been shown to be successful for anomaly detection by Kwon et al. (2020) [93]. They argued that "abnormal samples" cause more extreme changes to the model weights during training than regular instances, since the model has not learned them yet. Therefore, anomalies can be identified by their gradients [93]. This line of thought has parallels to our approach. By keeping only samples whose gradients align with the weight update computed on a large portion of the data, instances with atypical gradients are filtered out. These samples could be seen as outliers with respect to the comparison batch at the current model state.

6.3. Future Work

While the experiments in this thesis give some insights into potential advantages and pitfalls of the proposed method, the number of considered datasets is too low to draw any definitive conclusions. Moreover, only binary classification tasks were examined in this thesis, however, the method should be applicable to the multi-class setting without further modifications. Exploring the behaviour of the method when applied to a multi-class problem could provide some further intuition regarding the approach.

Furthermore, in our experiments, the denoising method was only used to train the classification layer. Directly incorporating the denoising method into the fine-tuning process of more complex models, such as CNNs or BERT, could potentially make the method more competitive. In theory, that should be possible, at least for the single-label

case, since gradient similarities were already computed for CNNs by Shi et al. (2021) [14]. The multi-label version of our method, as described by Algorithm 2, might need to be slightly adapted when learnable hidden layers are added, since the weights corresponding to each class are not clearly separated anymore, except for the weights of the final classification layer. One possible solution could be ignoring the entire sample instead of individual labels, like in the single-label case. Alternatively, the gradient comparison could be performed solely with respect to the output layer, like in the current formulation.

Lastly, it would be beneficial to directly compare the GM algorithm to related approaches, such as Cleanlab [10] or the *gambler's loss* by Ziyin et al. (2020) [36]. Cleanlab has a similar goal to our method, namely cleaning the training data, but tackles the task fairly differently. While Cleanlab removes likely mislabeled samples *before* the training process of the final model is started, we attempt to exclude or correct mislabeled samples *during* training. The approach by Ziyin et al. (2020) [36] has strong parallels to our method, since it also aims to dynamically identify labeling errors while the model is trained. In particular, it would be interesting to see whether samples that get assigned high uncertainty scores by their method are filtered out or relabeled by ours.

To conclude, the performance gains observed when applying the GM algorithm were likely not due to the method filtering out or relabeling mislabeled samples in particular. Investigating the method on more datasets might reveal the true cause of the improved scores.

Bibliography

[1] F. R. Cordeiro and G. Carneiro, "A survey on deep learning with noisy labels: How to train your model when you cannot trust on the annotations?," in *2020 33rd SIBGRAPI conference on graphics, patterns and images (SIBGRAPI)*, pp. 9–16, IEEE, 2020.

[2] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel: Rapid training data creation with weak supervision," *The VLDB Journal*, vol. 29, no. 2, pp. 709–730, 2020.

[3] A. J. Ratner, C. M. De Sa, S. Wu, D. Selsam, and C. Ré, "Data programming: Creating large training sets, quickly," *Advances in neural information processing systems*, vol. 29, 2016.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[5] C. Sun, A. Shrivastava, S. Singh, and A. Gupta, "Revisiting unreasonable effectiveness of data in deep learning era," in *Proceedings of the IEEE international conference on computer vision*, pp. 843–852, 2017.

[6] A. Sedova, A. Stephan, M. Speranskaya, and B. Roth, "Knodle: Modular weakly supervised learning with PyTorch," in *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, Association for Computational Linguistics, 2021.

[7] J. Zhang, Y. Yu, Y. Li, Y. Wang, Y. Yang, M. Yang, and A. Ratner, "WRENCH: A comprehensive benchmark for weak supervision," in *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2021.

[8] B. Frénay and M. Verleysen, "Classification in the presence of label noise: a survey," *IEEE transactions on neural networks and learning systems*, vol. 25, no. 5, pp. 845–869, 2014.

[9] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[10] C. Northcutt, L. Jiang, and I. Chuang, "Confident learning: Estimating uncertainty in dataset labels," *Journal of Artificial Intelligence Research*, vol. 70, pp. 1373–1411, 2021.

[11] C. G. Northcutt, A. Athalye, and J. Mueller, "Pervasive label errors in test sets destabilize machine learning benchmarks," *arXiv preprint arXiv:2103.14749*, 2021.

[12] G. Algan and I. Ulusoy, "Label noise types and their effects on deep learning," *arXiv preprint arXiv:2003.10471*, 2020.

[13] D. I. Inouye, P. Ravikumar, P. Das, and A. Dutta, "Hyperparameter selection under localized label noise via corrupt validation," in *NIPS Workshop*, 2017.

[14] Y. Shi, J. Seely, P. H. Torr, N. Siddharth, A. Hannun, N. Usunier, and G. Synnaeve, "Gradient matching for domain generalization," *arXiv preprint arXiv:2104.09937*, 2021.

[15] B. Zhao, K. R. Mopuri, and H. Bilen, "Dataset condensation with gradient matching," in *International Conference on Learning Representations*, 2021.

[16] T. A. Almeida, J. M. G. Hidalgo, and A. Yamakami, "Contributions to the study of sms spam filtering: new collection and results," in *Proceedings of the 11th ACM symposium on Document engineering*, pp. 259–262, 2011.

[17] A. Awasthi, S. Ghosh, R. Goyal, and S. Sarawagi, "Learning from rules generalizing labeled exemplars," in *International Conference on Learning Representations*, 2020.

[18] T. C. Alberto, J. V. Lochter, and T. A. Almeida, "Tubespam: Comment spam filtering on youtube," in *2015 IEEE 14th international conference on machine learning and applications (ICMLA)*, pp. 138–143, IEEE, 2015.

[19] G. Bénédict, V. Koops, D. Odijk, and M. de Rijke, "sigmoidf1: A smooth f1 score surrogate loss for multilabel classification," *arXiv preprint arXiv:2108.10566*, 2021.

[20] J. Irvin, P. Rajpurkar, M. Ko, Y. Yu, S. Ciurea-Ilcus, C. Chute, H. Marklund, B. Haghgoo, R. Ball, K. Shpanskaya, *et al.*, "Chexpert: A large chest radiograph dataset with uncertainty labels and expert comparison," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 33, pp. 590–597, 2019.

[21] C.-M. Teng, "Correcting noisy data.," in *ICML*, pp. 239–248, Citeseer, 1999.

[22] C. E. Brodley and M. A. Friedl, "Identifying mislabeled training data," *Journal of artificial intelligence research*, vol. 11, pp. 131–167, 1999.

[23] K. H. Le, T. V. Tran, H. H. Pham, H. T. Nguyen, T. T. Le, and H. Q. Nguyen, "Learning from multiple expert annotators for enhancing anomaly detection in medical image analysis," *arXiv preprint arXiv:2203.10611*, 2022.

[24] A. M. Davani, M. Díaz, and V. Prabhakaran, "Dealing with disagreements: Looking beyond the majority vote in subjective annotations," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 92–110, 2022.

[25] D. Karimi, H. Dou, S. K. Warfield, and A. Gholipour, "Deep learning with noisy labels: Exploring techniques and remedies in medical image analysis," *Medical Image Analysis*, vol. 65, p. 101759, 2020.

[26] D. Odaibo, Z. Zhang, F. Skidmore, and M. Tanik, "Detection of visual signals for pneumonia in chest radiographs using weak supervision," in *2019 SoutheastCon*, pp. 1–5, IEEE, 2019.

[27] D. Rolnick, A. Veit, S. Belongie, and N. Shavit, "Deep learning is robust to massive label noise," *arXiv preprint arXiv:1705.10694*, 2017.

[28] J. Ding, X. Li, and V. N. Gudivada, "Augmentation and evaluation of training data for deep learning," in *2017 IEEE international conference on big data (big data)*, pp. 2603–2611, IEEE, 2017.

[29] A. Ratner, S. H. Bach, H. Ehrenberg, J. Fries, S. Wu, and C. Ré, "Snorkel intro tutorial: Data labeling." `https://www.snorkel.org/use-cases/01-spam-tutorial`, 2020. (accessed: June 20, 2022).

[30] D. Cheng, T. Liu, Y. Ning, N. Wang, B. Han, G. Niu, X. Gao, and M. Sugiyama, "Instance-dependent label-noise learning with manifold-regularized transition matrix estimation," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 16630–16639, 2022.

[31] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[32] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[33] A. Ghosh, H. Kumar, and P. S. Sastry, "Robust loss functions under label noise for deep neural networks," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[34] Y. Wang, X. Ma, Z. Chen, Y. Luo, J. Yi, and J. Bailey, "Symmetric cross entropy for robust learning with noisy labels," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 322–330, 2019.

[35] S. Thulasidasan, T. Bhattacharya, J. Bilmes, G. Chennupati, and J. Mohd-Yusof, "Combating label noise in deep learning using abstention," *arXiv preprint arXiv:1905.10964*, 2019.

[36] L. Ziyin, B. Chen, R. Wang, P. P. Liang, R. Salakhutdinov, L.-P. Morency, and M. Ueda, "Learning not to learn in the presence of noisy labels," *arXiv preprint arXiv:2002.06541*, 2020.

[37] S. Sukhbaatar, J. Bruna, M. Paluri, L. Bourdev, and R. Fergus, "Training convolutional networks with noisy labels," *arXiv preprint arXiv:1406.2080*, 2014.

[38] B. Han, Q. Yao, T. Liu, G. Niu, I. W. Tsang, J. T. Kwok, and M. Sugiyama, "A survey of label-noise representation learning: Past, present and future," *arXiv preprint arXiv:2011.04406*, 2020.

[39] G. Patrini, A. Rozza, A. Krishna Menon, R. Nock, and L. Qu, "Making deep neural networks robust to label noise: A loss correction approach," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1944–1952, 2017.

[40] D. Hendrycks, M. Mazeika, D. Wilson, and K. Gimpel, "Using trusted data to train deep networks on labels corrupted by severe noise," *Advances in neural information processing systems*, vol. 31, 2018.

[41] J. Li, Y. Wong, Q. Zhao, and M. S. Kankanhalli, "Learning to learn from noisy labeled data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5051–5059, 2019.

[42] D. T. Nguyen, C. K. Mummadi, T. P. N. Ngo, T. H. P. Nguyen, L. Beggel, and T. Brox, "Self: Learning to filter noisy labels with self-ensembling," *arXiv preprint arXiv:1910.01842*, 2019.

[43] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," in *International conference on machine learning*, pp. 4334–4343, PMLR, 2018.

[44] P. W. Koh, S. Sagawa, H. Marklund, S. M. Xie, M. Zhang, A. Balsubramani, W. Hu, M. Yasunaga, R. L. Phillips, I. Gao, *et al.*, "Wilds: A benchmark of in-the-wild distribution shifts," in *International Conference on Machine Learning*, pp. 5637–5664, PMLR, 2021.

[45] P. Bandi, O. Geessink, Q. Manson, M. Van Dijk, M. Balkenhol, M. Hermsen, B. E. Bejnordi, B. Lee, K. Paeng, A. Zhong, *et al.*, "From detection of individual metastases to classification of lymph node status at the patient level: the camelyon17 challenge," *IEEE transactions on medical imaging*, vol. 38, no. 2, pp. 550–560, 2018.

[46] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading digits in natural images with unsupervised feature learning," 2011.

[47] W. Zeng, Y. Lin, Z. Liu, and M. Sun, "Incorporating relation paths in neural relation extraction," *arXiv preprint arXiv:1609.07479*, 2016.

[48] Y. Yu, S. Zuo, H. Jiang, W. Ren, T. Zhao, and C. Zhang, "Fine-tuning pre-trained language model with weak supervision: A contrastive-regularized self-training approach," *arXiv preprint arXiv:2010.07835*, 2020.

[49] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *International Journal of Data Warehousing and Mining (IJDWM)*, vol. 3, no. 3, pp. 1–13, 2007.

[50] W.-T. Chu and H.-J. Guo, "Movie genre classification based on poster images with deep neural networks," in *Proceedings of the Workshop on Multimodal Understanding of Social, Affective and Subjective Attributes*, pp. 39–45, 2017.

[51] J. Zhang, "Dataset: Format and usage." `https://github.com/JieyuZ2/wrench/wiki/Dataset:-Format-and-Usage`, 2021. (accessed: July 5, 2022).

[52] Y. Luan, J. Eisenstein, K. Toutanova, and M. Collins, "Sparse, dense, and attentional representations for text retrieval," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 329–345, 2021.

[53] A. Subakti, H. Murfi, and N. Hariadi, "The performance of bert as data representation of text clustering," *Journal of big Data*, vol. 9, no. 1, pp. 1–21, 2022.

[54] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*, vol. 39. Cambridge University Press Cambridge, 2008.

[55] D. Dessi, R. Helaoui, V. Kumar, D. R. Recupero, and D. Riboni, "Tf-idf vs word embeddings for morbidity identification in clinical notes: An initial study," *arXiv preprint arXiv:2105.09632*, 2021.

[56] Z. Yun-tao, G. Ling, and W. Yong-cheng, "An improved tf-idf approach for text classification," *Journal of Zhejiang University-Science A*, vol. 6, no. 1, pp. 49–55, 2005.

[57] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[58] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "Glue: A multi-task benchmark and analysis platform for natural language understanding," *arXiv preprint arXiv:1804.07461*, 2018.

[59] Y. Zhu, R. Kiros, R. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books," in *Proceedings of the IEEE international conference on computer vision*, pp. 19–27, 2015.

[60] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[61] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, *et al.*, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 conference on empirical methods in natural language processing: system demonstrations*, pp. 38–45, 2020.

[62] W. Samek, A. Binder, G. Montavon, S. Lapuschkin, and K.-R. Müller, "Evaluating the visualization of what a deep neural network has learned," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 11, pp. 2660–2673, 2016.

[63] S. Sharma, S. Sharma, and A. Athaiya, "Activation functions in neural networks," *towards data science*, vol. 6, no. 12, pp. 310–316, 2017.

[64] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[65] A. Yeh, "More accurate tests for the statistical significance of result differences," *arXiv preprint cs/0008005*, 2000.

[66] A. Kaur and R. Kumar, "Comparative analysis of parametric and non-parametric tests," *Journal of computer and mathematical sciences*, vol. 6, no. 6, pp. 336–342, 2015.

[67] S. L. Salzberg, "On comparing classifiers: Pitfalls to avoid and a recommended approach," *Data mining and knowledge discovery*, vol. 1, no. 3, pp. 317–328, 1997.

[68] E. W. Noreen, *Computer-intensive methods for testing hypotheses*. Wiley New York, 1989.

[69] L. Li, M. Doroslovački, and M. H. Loew, "Approximating the gradient of cross-entropy loss function," *IEEE Access*, vol. 8, pp. 111626–111635, 2020.

[70] Z. C. Lipton, C. Elkan, and B. Naryanaswamy, "Optimal thresholding of classifiers to maximize f1 measure," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 225–239, Springer, 2014.

[71] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, no. 1, pp. 1–13, 2020.

[72] S.-J. Yen and Y.-S. Lee, "Under-sampling approaches for improving prediction of the minority class in an imbalanced dataset," in *Intelligent Control and Automation*, pp. 731–740, Springer, 2006.

[73] D. Koleczek, A. Scarlatos, S. Karakare, and P. L. Pereira, "Umass pcl at semeval-2022 task 4: Pre-trained language model ensembles for detecting patronizing and condescending language," *arXiv preprint arXiv:2204.08304*, 2022.

[74] A. Maiza, "The unknown benefits of using a soft-f1 loss in classification systems." `https://towardsdatascience.com/the-unknown-benefits-of-using-a-soft-f1-loss-in-classification-systems-753902c0105d`, 2019. (accessed: June 30, 2022).

[75] K. Cao, C. Wei, A. Gaidon, N. Arechiga, and T. Ma, "Learning imbalanced datasets with label-distribution-aware margin loss," *Advances in neural information processing systems*, vol. 32, 2019.

[76] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, "Understanding deep learning (still) requires rethinking generalization," *Communications of the ACM*, vol. 64, no. 3, pp. 107–115, 2021.

[77] C. Garbin, P. Rajpurkar, J. Irvin, M. P. Lungren, and O. Marques, "Structured dataset documentation: a datasheet for chexpert," *arXiv preprint arXiv:2105.03020*, 2021.

[78] E. Giacomello, P. L. Lanzi, D. Loiacono, and L. Nassano, "Image embedding and model ensembling for automated chest x-ray interpretation," in *2021 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2021.

[79] Z. H. Hoo, J. Candlish, and D. Teare, "What is an roc curve?," 2017.

[80] X.-Z. Wu and Z.-H. Zhou, "A unified view of multi-label performance measures," in *International Conference on Machine Learning*, pp. 3780–3788, PMLR, 2017.

[81] I. Pillai, G. Fumera, and F. Roli, "F-measure optimisation in multi-label classifiers," in *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, pp. 2424–2427, IEEE, 2012.

[82] I. Pillai, G. Fumera, and F. Roli, "Threshold optimisation for multi-label classifiers," *Pattern Recognition*, vol. 46, no. 7, pp. 2055–2065, 2013.

[83] M. Tan and Q. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[84] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[85] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[86] A. Ke, W. Ellsworth, O. Banerjee, A. Y. Ng, and P. Rajpurkar, "Chextransfer: performance and parameter efficiency of imagenet models for chest x-ray interpretation," in *Proceedings of the Conference on Health, Inference, and Learning*, pp. 116–124, 2021.

[87] N. Inkawhich, "Finetuning torchvision models." `https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html`, 2017. (accessed: June 13, 2022).

[88] M. Karpinski and A. Macintyre, "Polynomial bounds for vc dimension of sigmoidal and general pfaffian neural networks," *Journal of Computer and System Sciences*, vol. 54, no. 1, pp. 169–176, 1997.

[89] A. K. Menon, A. S. Rawat, S. Reddi, and S. Kumar, "Multilabel reductions: what is my loss optimising?," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[90] K. Basterretxea, J. M. Tarela, and I. del Campo, "Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons," *IEE Proceedings-Circuits, Devices and Systems*, vol. 151, no. 1, pp. 18–24, 2004.

[91] R.-E. Fan and C.-J. Lin, "A study on threshold selection for multi-label classification," *Department of Computer Science, National Taiwan University*, pp. 1–23, 2007.

[92] H. Wei and N. Jethani, "Chexpertnet," *Journal of Machine Learning Research*, vol. 1, pp. 1–6, 2019.

[93] G. Kwon, M. Prabhushankar, D. Temel, and G. AlRegib, "Backpropagated gradient representations for anomaly detection," in *European Conference on Computer Vision*, pp. 206–226, Springer, 2020.

A. Appendix

A.1. Explicit Gradient Formulas

Since the gradient computation is an integral part of the GM algorithm, the theoretical gradient formulas for the investigated loss functions in this thesis are given in the following. Note that some of them were already mentioned in the main part of the thesis to gain a better understanding for the label changes made by the method. The gradients are given per sample. For the cross-entropy-based losses, the aggregated gradient is the mean of the individual gradients, while for the F_1 losses, the gradients are accumulated by taking the sum. The same notation as in the main part of the thesis is used. The F_1-based losses were inspired by Bénédict et al. (2021) [19] and Ashref Maiza (2019) [74].

A.1.1. Single-label Losses

Remember that in the main part of the thesis, the model outputs were denoted by the following expressions.

$$z_{t,1} = \sum_{f=1}^{F} (w_{f,1}\, x_{t,f}) + b_1$$

$$z_{t,2} = \sum_{f=1}^{F} (w_{f,2}\, x_{t,f}) + b_2$$

Moreover, note that the general derivative of the softmax function is given by the following expression (cf. Wang et al. 2019) [34].

$$\frac{\partial s_{t,k}}{\partial z_{t,j}} = \begin{cases} s_{t,k}(1 - s_{t,k}),\ j = k \\ -s_{t,k} s_{t,j},\ j \neq k \end{cases}$$

With this information, one can determine the sample-wise gradients of all single-label loss functions used throughout the thesis.

CE Loss

$$\mathcal{L}_{CE}(\mathcal{B}) = \frac{1}{M}\sum_{t=1}^{M} l_t$$

where

$$l_t = -[\log(s_{t,2})y_t + \log(s_{t,1})(1-y_t)] = -[\log(s_{t,2})y_t + \log(1-s_{t,2})(1-y_t)]$$

$$\begin{aligned}
\frac{\partial l_t}{\partial w_{f,1}} &= (1-s_{t,2})x_{t,f}y_t - s_{t,2}x_{t,f}(1-y_t) \\
\frac{\partial l_t}{\partial w_{f,2}} &= (s_{t,2}-1)x_{t,f}y_t + s_{t,2}x_{t,f}(1-y_t) \\
\frac{\partial l_t}{\partial b_1} &= (1-s_{t,2})y_t - s_{t,2}(1-y_t) \\
\frac{\partial l_t}{\partial b_2} &= (s_{t,2}-1)y_t + s_{t,2}(1-y_t)
\end{aligned}$$

F_1 Loss (Positive Class)

$$\mathcal{L}_{F_{1_{pos.}}}(\mathcal{B}) = 1 - \frac{2\widehat{tp}}{2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon}$$

$$\begin{aligned}
\left(\frac{\partial \mathcal{L}_{F_{1_{pos.}}}(\mathcal{B})}{\partial w_{f,1}}\right)_t &= 2\Big(\frac{y_t s_{t,2}(1-s_{t,2})x_{t,f}(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&\quad - \frac{(y_t s_{t,2}(1-s_{t,2})x_{t,f} + (1-y_t)s_{t,2}(1-s_{t,2})x_{t,f})\widehat{tp}}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\Big) \\
\left(\frac{\partial \mathcal{L}_{F_{1_{pos.}}}(\mathcal{B})}{\partial w_{f,2}}\right)_t &= -2\Big(\frac{y_t s_{t,2}(1-s_{t,2})x_{t,f}(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&\quad - \frac{(y_t s_{t,2}(1-s_{t,2})x_{t,f} + (1-y_t)s_{t,2}(1-s_{t,2})x_{t,f})\widehat{tp}}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\Big) \\
\left(\frac{\partial \mathcal{L}_{F_{1_{pos.}}}(\mathcal{B})}{\partial b_1}\right)_t &= 2\Big(\frac{y_t s_{t,2}(1-s_{t,2})(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&\quad - \frac{(y_t s_{t,2}(1-s_{t,2}) + (1-y_t)s_{t,2}(1-s_{t,2}))\widehat{tp}}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\Big) \\
\left(\frac{\partial \mathcal{L}_{F_{1_{pos.}}}(\mathcal{B})}{\partial b_2}\right)_t &= -2\Big(\frac{y_t s_{t,2}(1-s_{t,2})(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&\quad - \frac{(y_t s_{t,2}(1-s_{t,2}) + (1-y_t)s_{t,2}(1-s_{t,2}))\widehat{tp}}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\Big)
\end{aligned}$$

Macro-F_1 Loss

$$\mathcal{L}_{F_{1_{macro}}}(\mathcal{B}) = 1 - \frac{1}{2}[\frac{2\widehat{tp}}{2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon} + \frac{2\widehat{tn}}{2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon}]$$

$$\begin{aligned}
\left(\frac{\partial \mathcal{L}_{F_{1_{macro}}}(\mathcal{B})}{\partial w_{f,1}}\right)_t &= \frac{1}{2}[\left(\frac{\partial \mathcal{L}_{F_{1_{pos.}}}(\mathcal{B})}{\partial w_{f,1}}\right)_t + \\
&+ 2\big(\frac{-(1-y_t)s_{t,2}(1-s_{t,2})x_{t,f}(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&- \frac{(-(1-y_t)(1-s_{t,2})x_{t,f} - y_t s_{t,2}(1-s_{t,2})x_{t,f})\widehat{tn}}{(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\big)] \\
\left(\frac{\partial \mathcal{L}_{F_{1_{macro}}}(\mathcal{B})}{\partial w_{f,2}}\right)_t &= \frac{1}{2}[\left(\frac{\partial \mathcal{L}_{F_{1_{pos.}}}(\mathcal{B})}{\partial w_{f,2}}\right)_t - \\
&- 2\big(\frac{-(1-y_t)s_{t,2}(1-s_{t,2})x_{t,f}(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&- \frac{(-(1-y_t)(1-s_{t,2})x_{t,f} - y_t s_{t,2}(1-s_{t,2})x_{t,f})\widehat{tn}}{(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\big)] \\
\left(\frac{\partial \mathcal{L}_{F_{1_{macro}}}(\mathcal{B})}{\partial b_1}\right)_t &= \frac{1}{2}[\left(\frac{\partial \mathcal{L}_{F_{1_{pos.}}}(\mathcal{B})}{\partial b_1}\right)_t + \\
&+ 2\big(\frac{-(1-y_t)s_{t,2}(1-s_{t,2})(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&- \frac{(-(1-y_t)(1-s_{t,2}) - y_t s_{t,2}(1-s_{t,2}))\widehat{tn}}{(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\big)] \\
\left(\frac{\partial \mathcal{L}_{F_{1_{macro}}}(\mathcal{B})}{\partial b_2}\right)_t &= \frac{1}{2}[\left(\frac{\partial \mathcal{L}_{F_{1_{pos.}}}(\mathcal{B})}{\partial b_2}\right)_t - \\
&- 2\big(\frac{-(1-y_t)s_{t,2}(1-s_{t,2})(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&- \frac{(-(1-y_t)(1-s_{t,2}) - y_t s_{t,2}(1-s_{t,2}))\widehat{tn}}{(2\widehat{tn} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\big)]
\end{aligned}$$

A.1.2. Multi-Label Losses

The multi-label losses examined in this thesis use a sigmoid activation. The derivative of this function is given by [90]:

$$\frac{\partial \sigma(z_{t,k})}{\partial z_{t,k}} = \sigma(z_{t,k})(1 - \sigma(z_{t,k}))$$

BCE Loss

$$\mathcal{L}^*{}_{BCE} = \frac{1}{MK}\sum_{t=1}^{M}\sum_{k=1}^{K} l^*_{t,k}$$

where

$$l^*_{t,k} = -[\log(\sigma(z_{t,k}))y_{t,k} + \log(1-\sigma(z_{t,k}))(1-y_{t,k})]$$

$$\frac{\partial l^*_{t,k}}{\partial w_{f,k}} = (\sigma(z_{t,k})-1)x_{t,f}y_{t,k} + \sigma(z_{t,k})x_{t,f}(1-y_{t,k})$$
$$\frac{\partial l^*_{t,k}}{\partial b_k} = (\sigma(z_{t,k})-1)y_{t,k} + \sigma(z_{t,k})(1-y_{t,k})$$

Macro-F_1 Loss

$$\mathcal{L}^*{}_{F_{1macro}}(\mathcal{B}) = 1 - \frac{1}{K}\sum_{k=1}^{K}\frac{2\widetilde{tp}_k}{2\widetilde{tp}_k + \widetilde{fn}_k + \widetilde{fp}_k + \epsilon}$$

$$\left(\frac{\partial \mathcal{L}^*{}_{F_{1macro}}(\mathcal{B})}{\partial w_{f,k}}\right)_t = -\frac{2}{K}\Big(\frac{y_{t,k}\sigma(z_{t,k})(1-\sigma(z_{t,k}))x_{t,f}(2\widehat{tp}_k + \widehat{fp}_k + \widehat{fn}_k + \epsilon)}{(2\widehat{tp}_k + \widehat{fp}_k + \widehat{fn}_k + \epsilon)^2} -$$
$$- \frac{(y_t\sigma(z_{t,k})(1-\sigma(z_{t,k}))x_{t,f} + (1-y_{t,k})\sigma(z_{t,k})(1-\sigma(z_{t,k}))x_{t,f})\widehat{tp}_k}{(2\widehat{tp}_k + \widehat{fp}_k + \widehat{fn}_k + \epsilon)^2}\Big)$$

$$\left(\frac{\partial \mathcal{L}^*{}_{F_{1macro}}(\mathcal{B})}{\partial b_k}\right)_t = -\frac{2}{K}\Big(\frac{y_{t,k}\sigma(z_{t,k})(1-\sigma(z_{t,k}))(2\widehat{tp}_k + \widehat{fp}_k + \widehat{fn}_k + \epsilon)}{(2\widehat{tp}_k + \widehat{fp}_k + \widehat{fn}_k + \epsilon} -$$
$$- \frac{(y_t\sigma(z_{t,k})(1-\sigma(z_{t,k})) + (1-y_{t,k})\sigma(z_{t,k})(1-\sigma(z_{t,k})))\widehat{tp}_k}{(2\widehat{tp}_k + \widehat{fp}_k + \widehat{fn}_k + \epsilon)^2}\Big)$$

Micro-F_1 Loss

$$\mathcal{L}^*_{F_{1_{micro}}}(\mathcal{B}) = 1 - \frac{2\widetilde{tp}}{2\widetilde{tp} + \widetilde{fn} + \widetilde{fp} + \epsilon}$$

$$\begin{aligned}
\left(\frac{\partial \mathcal{L}^*_{F_{1_{micro}}}(\mathcal{B})}{\partial w_{f,k}}\right)_t &= -\frac{2}{K}\Big(\frac{y_{t,k}\sigma(z_{t,k})(1-\sigma(z_{t,k}))x_{t,f}(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&\quad - \frac{(y_t\sigma(z_{t,k})(1-\sigma(z_{t,k}))x_{t,f} + (1-y_{t,k})\sigma(z_{t,k})(1-\sigma(z_{t,k}))x_{t,f})\widehat{tp}}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\Big) \\
\left(\frac{\partial \mathcal{L}^*_{F_{1_{micro}}}(\mathcal{B})}{\partial b_k}\right)_t &= -\frac{2}{K}\Big(\frac{y_{t,k}\sigma(z_{t,k})(1-\sigma(z_{t,k}))(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2} - \\
&\quad - \frac{(y_t\sigma(z_{t,k})(1-\sigma(z_{t,k})) + (1-y_{t,k})\sigma(z_{t,k})(1-\sigma(z_{t,k})))\widehat{tp}}{(2\widehat{tp} + \widehat{fp} + \widehat{fn} + \epsilon)^2}\Big)
\end{aligned}$$

A.2. CheXpert Development Performance

This section provides the central results on the development set created for CheXpert, which were omitted in the main part of the thesis due to space constraints.

A.2.1. BCE Loss

Method	Update/Comp. Loss	Threshold Tuning	Avg. Macro-F_1	Avg. Micro-F_1	Avg. AUROC
GM	BCE/BCE	Yes	0.52569(0.00018)	0.61748(0.00014)	0.77399(0.00067)
Baseline	BCE	Yes	**0.52685(0.000507)**	**0.61783(0.00102)**	**0.77471(0.00327)**

Table A.1.: Dev. Performance on CheXpert (BCE Loss, $\tau = 0$)

A.2.2. Macro-F_1 Loss

Method	Update/Comp. Loss	Threshold Tuning	Avg. Macro-F_1	Avg. Micro-F_1	Avg. AUROC
GM	Macro-F_1/Macro-F_1	No	0.50823(0.0041)	0.53927(0.00078)	0.73917(0.00073)
GM	Macro-F_1/Macro-F_1	Yes	0.51799(0.00146)	0.60082(0.00034)	0.73917(0.00073)
GM	BCE/Macro-F_1	Yes	0.52689(0.00018)	**0.61649(0.00038)**	0.77274(0.0003)
Baseline	Macro-F_1	No	0.52709(0.00038)	0.59024(0.00222)	**0.77427(0.00019)**
Baseline	Macro-F_1	Yes	**0.52855(0.00043)**	0.61409(0.00102)	**0.77427(0.00019)**

Table A.2.: Dev. Performance on CheXpert (Macro-F_1 Loss, $\tau = 0$)

A.2.3. Micro-F_1 Loss

Method	Update/Comp. Loss	Threshold Tuning	Avg. Macro-F_1	Avg. Micro-F_1	Avg. AUROC
GM	Micro-F_1/Micro-F_1	No	0.44321(0.002461)	0.60384(0.00034)	0.71781(0.00117)
GM	Micro-F_1/Micro-F_1	Yes	0.49529(0.0007)	0.60484(0.00038)	0.71781(0.00117)
GM	BCE/Micro-F_1	Yes	**0.52366(0.00245)**	0.61794(0.00037)	**0.77229(0.00025)**
Baseline	Micro-F_1	No	0.49024(0.00143)	0.61747(0.00051)	0.77025(0.00022)
Baseline	Micro-F_1	Yes	0.51621(0.00135)	**0.61888(0.00031)**	0.77025(0.00022)

Table A.3.: Dev. Performance on CheXpert (Micro-F_1 Loss, $\tau = 0$)