CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Distributed Semi-Supervised Learning and Audio Recognition of Road Surfaces

Master's thesis in Computer science and engineering

Adam Davidsson
Simon Larsson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

Distributed Semi-Supervised Learning and Audio Recognition of Road Surfaces

Adam Davidsson
Simon Larsson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Distributed Semi-Supervised Learning and Audio Recognition of Road Surfaces
Adam Davidsson
Simon Larsson

Supervisor: Yehia Abd Alrahman, Computer Science and Engineering
Advisor: Tomas Carlfalk, WirelessCar
Examiner: Krasimir Angelov, Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Gothenburg, Sweden 2022

Distributed Semi-Supervised Learning and Audio Recognition of Road Surfaces
Adam Davidsson
Simon Larsson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The automotive industry is a promising environment for machine learning. However, current machine learning techniques do not meet all the requirements of many possible applications. Requirement such as privacy preservation, limited communication and semi-supervision. To satisfy these requirements, this thesis proposes a simple distributed semi-supervised algorithm (distributed FixMatch). Furthermore, we apply this algorithm to a real-world problem, detecting road surface types from audio. In applying the semi-supervised algorithm to this problem, we also propose a simple augmentation technique for audio features. The proposed algorithm was tested on two real datasets, where the algorithm was compared to a supervised training algorithm. The results suggest that the algorithm successfully leveraged unlabeled data. Furthermore, a theoretical analysis and a simulation show that the communication cost of the proposed algorithm was lower than federated or centralized alternatives.

Keywords: Machine learning, Federated learning, Semi-Supervised learning, Distributed learning, Audio Recognition, Road Surface detection, Neural Network.

Acknowledgements

We want to thank our supervisor, Yehia Abd Alrahman for his contributions to this thesis. We also want to thank Jens Andersson, Tomas Carlfalk, Anna Gunlycke, and Jeffrey Spång at Wirelesscar for their help during the project. Finally, we want to thank Linn Alholt at New Minds for connecting us with WirelessCar.

Adam Davidsson, Gothenburg, June 2022
Simon Larsson, Gothenburg, June 2022

Contents

List of Figures **xi**

List of Tables **xiii**

1 Introduction **1**
1.1 Problem . . . 2
1.2 Our Contribution . . . 2
1.3 Related works . . . 3
1.3.1 Federated learning and distributed learning . . . 3
1.3.2 Semi-supervised learning . . . 4
1.3.3 Audio classifiers . . . 4
1.4 Overview . . . 5

2 Theory **7**
2.1 Audio recognition . . . 7
2.1.1 Signal processing & Feature extraction . . . 7
2.1.2 Artificial neural networks . . . 9
2.2 Federated learning . . . 12
2.3 Semi-supervised learning . . . 14
2.3.1 FixMatch . . . 17
2.4 Distributed algorithms . . . 19
2.4.1 Model of Communication . . . 20

3 Methods **23**
3.1 Data Gathering . . . 23
3.2 Datasets . . . 24
3.2.1 Road surface noise dataset . . . 24
3.2.2 UrbanSound8K . . . 25
3.3 Feature Extraction . . . 26
3.3.1 Auditory Spectral Features . . . 26
3.3.2 Feature set evaluation . . . 27
3.3.3 eGeMAPSv02 . . . 28
3.4 Model Architecture & Training . . . 29
3.4.1 Distributed FixMatch . . . 30
3.5 Hyperparameter optimization . . . 33
3.6 Simulation . . . 33
3.7 Evaluation . . . 34

4 Results **35**
4.1 Communication overhead 39

5 Concluding remarks **43**
5.1 Discussion 43
5.2 Conclusion 44
5.3 Future work 45

Bibliography **47**

A Appendix 1 **I**

List of Figures

2.1 A short audio recording of a wheel driving on asphalt, represented in the time domain (a) and in the frequency domain (b). 8
2.2 A fully connected neural network. 9
2.3 Two types of federated learning. 2.3b illustrates a bank and a shop using a coordinator to train together without revealing private information 13
2.4 A semi-supervised example. The stars represent labeled data. The black line represents the decision boundary for only training on the labeled data. The grey line represents the decision boundary for training on the labeled data and the unlabeled data 15
2.5 An example of consistency loss. The arrows represent data augmentations. Two random augmentations are applied to each data point. After the augmentations, the two new points should likely remain in the same class. 17
2.6 A diagram illustrating the difference between a centralized, decentralized, and distributed network topology. 20

3.1 The placement of the microphone when collecting road noise for the road surface noise dataset. 24
3.2 State transitions of distributed FixMatch 30
3.3 The distribution of feature mfcc1_sma3_amean for all road surface data points visualized for each class. 32

4.1 The accuracy difference of distributed FixMatch vs a central supervised model for the Road Surface dataset using a fixed number of labeled training examples for each class. 36
4.2 The accuracy difference of distributed FixMatch vs a central supervised model for the UrbanSound8K dataset using a fixed number of labeled training examples for each class. 36
4.3 The distribution of feature mfcc1_sma3_amean for all UrbanSound8K data points visualized for each class. 37
4.4 Accuracy development of the nodes during training on the UrbanSound8K dataset. 38
4.5 Accuracy of the different road surfaces. Trained using distributed FixMatch. Road surface noise accuracy was taken when a client reached at least 87% accuracy. An average of 5 runs was taken 39

4.6 The communication overhead of the most strained node for Distributed FixMatch vs Federated training 42

List of Tables

3.1 Parameters of the cars used for recording 23
3.2 Accuracy of different feature sets when used to train a neural network to classify the Road surface noise dataset 27
3.3 Accuracy of a distributed FixMatch when trained on the UrbanSound8K dataset for different combinations of hyperparameters. . . . 33

4.1 Number of clients affect on accuracy 38
4.2 Communication complexity. Where n is the number of nodes participating in the training, d is the client dataset size, r is the number of server rounds during training and m is the model size. 40
4.3 Number of communication messages sent for distributed FixMatch vs federated training until achieving a threshold accuracy 41

A.1 Accuracies depicted in Figure 4.2 and Figure 4.1 I

1

Introduction

Recent years have witnessed an increasing interest in machine learning. This trend has been especially strong in the automotive industry, with use cases such as autonomous vehicles and automatic braking systems. This trend can be shown by the fact that around 95% of all new vehicles in the next ten years may have some form of AI [18]. This is a significant increase compared to 2015, where the number was as low as 5-10%[18].

One of the potential use cases for machine learning in the automotive industry is road surface detection. The idea is to detect which type of road surface the vehicle is driving on, e.g., asphalt, gravel, or icy asphalt. In addition to detecting road surface, the roughness of different roads could be detected. Road surfaces and road roughness can be used to determine the friction forces between the road and the wheels, which is important when developing safety measures. Another use case could be the prediction of energy consumption in electric vehicles, which is also dependent on the road material [45].

Machine learning is a resource-intensive process. The effectiveness of a model largely depends on the amount and the quality of data used in the training process [20]. Massive amounts of data are constantly collected by vehicles. Modern vehicles have around 60-100 sensors and 30-50 computers [24]. However, sending massive amounts of data to a centralized location is expensive and scales badly[33]. One way to solve this problem is to distribute the learning process among different vehicles. A vehicle in a federated system could send generalized knowledge, instead of raw data, to a central server. The server can then share the knowledge with other vehicles. However, data would still be sent to a centralized location which could cause problems with scaling. The alternative would be a fully distributed system. Where every participant takes turns in acting as client and server. Two other concerns strengthen the case for a distributed system. Firstly, the collection of private data from vehicles. A 2019 survey found that 79% of Americans were concerned with how data was collected by companies [8]. This concern could be alleviated by only collecting the data important for learning instead of collecting all raw data. Secondly, renting or buying computers to train the model in a centralized way is expensive. One way to avoid this would be to use the already existing computers in vehicles.

1.1 Problem

Our research problem is focused on studying if federated machine learning and semi-supervised machine learning techniques can be adapted to the requirements of the automotive industry. We will focus on one automotive industry machine learning problem, classifying the road surface a car is driving on, through audio recognition.

Classifying road noise in the automotive setting comes with a few problems. Firstly, each vehicle has the possibility to gather large amounts of unlabeled data. Unlabeled data lacks any information describing it. In contrast, labeled data has been tagged with some information that explains it. For instance, an audio recording tagged with the information that it is a recording of a vehicle driving on asphalt. Labeling the data gathered by the cars is resource-intensive. Therefore the amount of labeled data is restricted. Labeled data is much more useful for training classification models. With labeled data, we can train a model to connect the features X to the Label Y. However, with unlabeled data we have no such labels to connect the features with. This makes classification harder. Secondly, the amount of information that can be communicated between the vehicles and a potential server is limited. The communication between the vehicles should be minimized since it is expensive. Furthermore, the amount of data a server can receive is limited because of bandwidth. This causes scaling problems when the number of vehicles and updates increases. The third problem is how to preserve privacy. Generalized knowledge should be extracted from the raw data without sending it to a central location, i.e., the data should be kept on the vehicle.

To address these problems, a distributed algorithm, based on previous federated machine learning and semi-supervised machine learning techniques, is proposed. Federated learning techniques allow for learning without sending raw data to a central location. Semi-supervised learning techniques allow for both unlabeled and labeled data to be used in training. These techniques are combined in a distributed algorithm. The distribution lowers the communication overhead.

1.2 Our Contribution

The high-level goal of this project is to determine whether federated learning can effectively be applied to problems in the automotive industry. To be useful to the automotive industry it needs to meet the following design requirements[33][8][24]:

- The communication costs between the participants (cars) should be minimized.
- The training needs to work on unlabeled data.
- The previous two goals need to be accomplished without loss of privacy or accuracy

To fulfill these goals this project will combine state-of-the-art federated learning, semi-supervised learning, and audio classification techniques. To detect the road surface, a neural network will be trained on features extracted from road noise. To

minimize communication costs, the training will be performed with our proposed distributed algorithm. The algorithm is based on federated learning techniques to avoid uploading personal data and semi-supervised learning to leverage unlabeled and labeled data. To our knowledge, we are the first to do this. So our contribution will be:

- Proposing a novel distributed algorithm that combines FixMatch (a state-of-the-art semi-supervised algorithm) and FedAvg (a state-of-the-art federated learning algorithm)
- Proposing a simple augmentation technique for audio features to be used in semi-supervised machine learning
- Classifying road surfaces based on road noise with a neural network

1.3 Related works

To solve the problems described in the previous section, techniques from federated learning, semi-supervised learning, and audio classification will be used. This section will introduce some previous work that has been done in these fields.

1.3.1 Federated learning and distributed learning

One proposed method for solving the problem of distributing the learning process is federated learning (FL) [28]. The method usually consists of loosely connected devices (called clients) and a central server. The server coordinates the sharing of knowledge between the clients. FedAvg [28] is one such federated learning method. In FedAvg, the server sends its model to the clients. The clients then train the model on their local data and send the resulting model back. The server then repeats the process with an average of all the models it received from the clients. FedAVG has been empirically shown to work. It could achieve 85% accuracy on the Cifar-10 dataset after 2000 communication rounds.

Another proposed algorithm for performing federated learning is FedMatch [22]. It operates much in the same way as FedAvg with clients training their own models and a central server aggregating the knowledge learned by the models. However, unlike traditional FL strategies, FedMatch can be used in scenarios where only a small amount of labeled data is available at the central server. The remaining data is unlabeled and distributed among many clients. The main ideas of FedMatch are parameter decomposition and inter-client consistency loss. FedMatch decomposes the parameters of the model to allow disjoint learning on the labeled and the unlabeled data. One set of parameters is trained on the labeled data and one set is trained on the unlabeled data. Separating the parameters prevents for loss of knowledge learned on the labeled data which improves accuracy [22]. Inter-client consistency loss regularizes the models learned at multiple clients to output the same prediction [22]. This allows for improved model consistency across the multiple models and improved accuracy.

Another approach was taken by [46]. They introduced a supervised algorithm (DADA) without a central server. Instead, the collaboration is done through peer-to-peer updates. Each peer exchanges updates with only a few neighbors. This exchange can be described by a collaboration graph. This graph is also optimized such that peers with similar tasks share knowledge with each other. The algorithm is based on Frank-Wolfe i.e. conditional gradient descent, and peer sampling service. Tests on synthetic data showed that the Dada successfully achieved high accuracy while keeping the collaboration graph sparse.

Several distributed semi-supervised algorithms have been developed [16][47][36]. However, these algorithms tend to be too complicated to implement [16][47]. Furthermore, they have been implemented with image recognition or general classification tasks in mind [16][47]. Additionally, one of the algorithms was proposed to solve the binary classification problem [36].

1.3.2 Semi-supervised learning

MixMatch was proposed to solve the challenge of using both labeled and unlabeled data to train an image classifier [7]. The method is based on three techniques: entropy minimization, which is used to make confident predictions on unlabeled data; consistency regularization, which means that the same output should be produced by the model when the input data is slightly altered; and generic regularization, which aims to generalize the model and avoid overfitting. The three techniques of MixMatch are built around image augmentation. The unlabeled data is augmented k times. The average class distribution of the k augmentations is then used as a label in training. Tests on well-known datasets showed that MixMatch achieved a significantly improved performance over other semi-supervised methods.

Another proposed method is Unsupervised Data Augmentation (UDA) [44]. UDA is very similar to MixMatch. It also uses image augmentation to perform semi-supervised learning. What differentiates UDA from MixMatch is the way data augmentation is performed. MixMatch only uses weak augmentation, both on labeled and unlabeled data. UDA, however, uses strong augmentation for the unlabeled data. The stronger data augmentations are used to get more diverse input data. UDA also introduces a classification threshold. This threshold is used to filter out unlabeled examples that the model can not classify with high confidence. UDA achieves great results for well-known image classification tasks.

1.3.3 Audio classifiers

Several different papers have been published in the field of audio classifiers. A paper published in 2013 introduces a method for classifying road surface wetness from audio [3]. The article extracts auditory spectral features by applying the short-time Fourier transform. The features are converted to Log-Mel spectrograms. These transformed features are then used to train a recurrent neural network (RNN). The network can predict road wetness with an Unweighted Average Recall of 93.2%

Another article published in 2021 uses a similar method of auditory feature extraction [39]. The paper introduces a method for training convolutional neural networks in the supervised federated setting. They evaluate their model on three different data sets: Speech Commands [43], Ambient Context, [32] and VoxForge [42]. They achieve an accuracy of 96.5%, 73.0%, and 79.6% respectively in the centralized setting. And 96.9%, 71.9%, and 79.1% in the federated setting.

A different classifier was tried by [4]. Their goal was to classify the road surface roughness using audio. They also used a short-time Fourier transform to extract auditory features and then converted them to Log-Mel spectrograms. However, instead of using an RNN for the classification, they used a Convolutional Neural Network (CNN). The trained model was able to classify the audio into two classes, rough and smooth, with an F1-score of 86%. This approach was further worked on [19]. Instead of using a CNN, a Siamese Neural Network (SNN) was used. The SNN was used since it has good generalization properties on unbalanced data. The new approach improved the performance of the classification to an F1-score of 95.6%.

An audio classifier was proposed by [31]. They used a semi-supervised algorithm called FixMatch [38] to perform the training. FixMatch was originally proposed to solve image recognition tasks. However, [31] evaluated the performance of FixMatch on audio recognition. To get FixMatch to work with audio signals, they converted the audio signals to spectrograms. The spectrograms were then used as input images to train a CNN using FixMatch. Their results showed that FixMatch doesn't necessarily work out-of-the-box on audio data.

1.4 Overview

In the theory chapter, we present the background knowledge needed to follow the report. We begin by explaining the audio recognition techniques. We then describe the federated learning knowledge needed to understand our algorithm. We also explain how semi-supervised learning works and present an algorithm that our implementation builds upon. Lastly, general distributed algorithms are explained to understand how an algorithm can be decentralized.

The methodology chapter describes the approach which is taken to fulfill the goal presented in the introduction chapter. First, we describe how road noise data will be gathered. Then, we present the feature extraction method and why it was chosen. The following section describes the method by which those features will be classified. The following two sections go into how we optimize and simulate the training. Lastly. we describe how the goals will be evaluated.

In the results chapter, we present the result of the evaluations. We demonstrate the trained classifier accuracy and how it compares to other approaches. We also evaluate the communication costs of our approach compared to alternative approaches.

The Discussion chapter mentions challenges we faced during the thesis project. We

also discuss the results and analyze what was achieved. Lastly, we mention future work that can be done to make improvements.

2

Theory

This section will give a detailed background to the techniques used to solve the problems described in the previous chapter. The first section will give a basic introduction to audio recognition and neural networks since it is crucial to understanding the evaluation method. The second section will give an introduction to federated learning. Additionally, we will explain different strategies to perform federated learning. We will also give a description of an algorithm, FedAvg, that will be used as part of our method. These three topics are important for understanding the federated component of distributed FixMatch and the context it was developed in. The third section will give an overview of different semi-supervised techniques. Furthermore, it will give a detailed description of the FixMatch algorithm. This section is also included to give context and basic background for the semi-supervised component of distributed FixMatch. The final section will give a short explanation of distributed methods to understand how a centralized algorithm can be distributed.

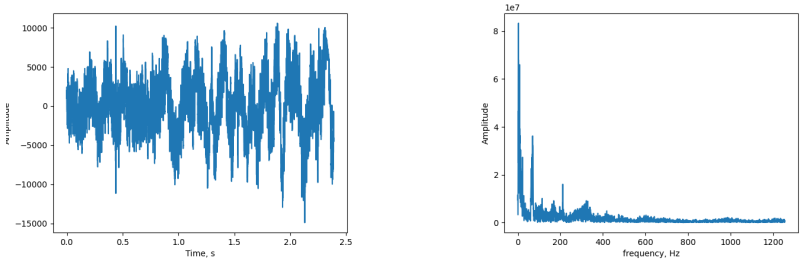
2.1 Audio recognition

Audio recognition is a technology used to identify and classify audio data. It can be used to classify speech, music, emotions, etc. There are many different approaches to classifying audio. However, the methods can usually be broken down into three steps: signal processing, feature extraction, and classification. Signal processing converts the input audio from the time domain into the frequency domain. Feature extraction extracts relevant feature vectors from the processed data. Lastly, a classifier uses the extracted features to make classifications.

2.1.1 Signal processing & Feature extraction

Signal processing and feature extraction is the process of extracting numerical features from audio signals. Audio signals come in two forms: analog and digital. Analog sound is a replica of the sound wave, while digital sound is built up from samples of the original sound wave.

Audio signals can be represented in two domains. In the time domain and in the frequency domain. The time domain shows the amplitude of the audio signal. With analog sound, it shows a continuous amplitude curve while digital sound shows the



(a) Time domain **(b)** Frequency domain

Figure 2.1: A short audio recording of a wheel driving on asphalt, represented in the time domain (a) and in the frequency domain (b).

amplitude at each sample point. In the frequency domain, the amplitude is shown along a frequency axis instead of the time axis. To transform the signal from the time domain to the frequency domain, a Fourier Transform is applied. Applying a Fourier Transform splits a signal into its constituent frequencies. These frequencies can then be shown in the frequency to amplitude representation. In Fig. 2.1a and Fig. 2.1b we show the differences between the two representations. In Fig. 2.1a, an audio recording of a wheel driving on asphalt is represented in the time domain. In Fig. 2.1b the same audio clip is represented but in the frequency domain.

From the time domain and the frequency domain, low-level descriptors (LLD) are extracted. Low-level descriptors are sound parameters and are closely related to the sound signal. Examples of LLD are pitch, loudness, and spectral slope. Low-level descriptors are represented as floating-point values in a 1-dimensional array. The LLD are fed to functions to obtain a single value from each LLD. Two examples of such functions are arithmetic mean and standard deviation. The function values of all LLD are then combined to form a resulting feature extraction set. To extract LLD and apply functionals, a toolkit called openSMILE [15] will be used.

Feature extraction is used to reduce the dimensionality of the input data, i.e., the number of features used by the model to make a prediction. Reducing the number of features has several benefits. First, it reduces the number of computations required in training and it can reduce the size of the data. Fewer computations mean faster training speed and therefore possibly better results. A second benefit is the removal of multicollinearity, i.e., several independent variables that are correlated, which can be a problem in regression models. However, multicollinearity is less of a problem for neural networks[41]. A third benefit of feature extraction is that it can help with the curse of dimensionality. The curse of dimensionality refers to various problems occurring when analyzing data with many dimensions, that do not occur in lower dimensions. One such problem is that more features require more training examples [9].

2.1.2 Artificial neural networks

Artificial neural networks (ANNs) consist of groups of connected nodes and weights. ANNs can act as functions and given a certain input they will produce a certain output. Furthermore, ANNs can be trained to produce a certain output, given a certain input, through the backpropagation algorithm.

Layers

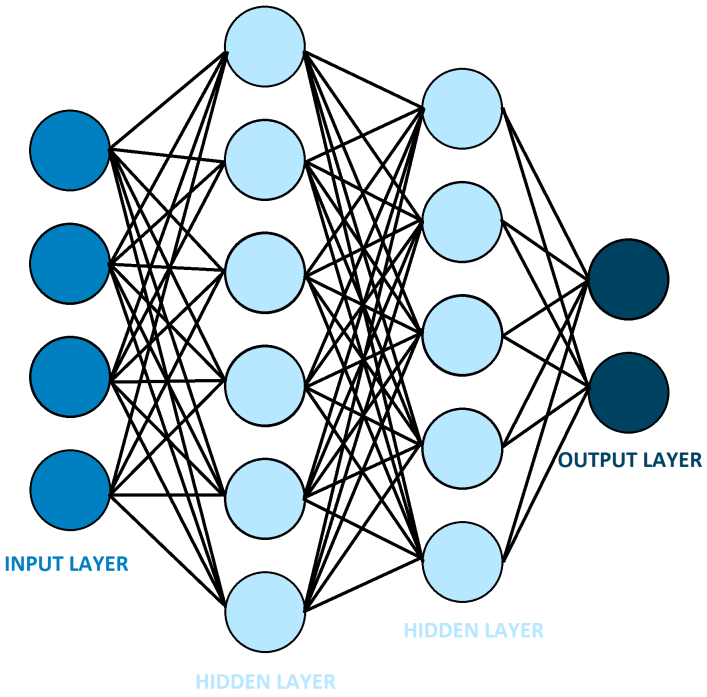


Figure 2.2: A fully connected neural network.

The nodes in ANNs are connected to each other through weights. The nodes are divided into different layers. There are three types of layers. The input layer, the hidden layers, and the output layer. The number of layers, weights, and nodes depends on the application. In a fully connected neural network, each node is connected with weights to all the nodes in the layer below it, as shown in Fig. 2.2 The basic idea of the ANN is that each node should activate on some input. When one node activates, it causes a node in a later layer to activate. This causes a hierarchical activation of nodes. For instance in the case of image classification. The nodes in the input layer may activate when a certain pixel has a specific value. A group of nodes in the input layer could represent an edge. When they activate they could cause a node in the next hidden layer to activate. The activated node could be part of a group that represents a dog-ear or a cat-eye. This group could cause a node in the output layer to activate. Which could Indicate if the image depicts a dog or a cat.

Gradient descent

ANNs are trained using gradient descent. The goal of gradient descent is to decrease a loss function. When the slope of the loss function is 0, the minimum loss has been found. Thus the gradient descent algorithm tries to step towards the minima of the loss function. To achieve this, firstly the loss function is applied to the model prediction and the real label. The loss function calculates some distance between the predicted and real values. A commonly used loss function is cross entropy loss. Cross entropy is defined as:

$$L_{CE} = -\sum_{i=1}^{n} y_i \log \hat{y}_i, \text{ for n classes,} \tag{2.1}$$

where y_i is the real value and $\hat{y}_i$ is the predicted value fed through an activation function. The activation function used with cross-entropy loss is usually softmax. Softmax transforms the output values to an array of probabilities. The probabilities indicate how likely the input is to belong to each class.

Secondly, to decrease the loss function, a gradient is calculated on it using backwards propagation. Finally, the parameters of the model is then updated with the negative gradient to minimize the loss function:

$$w_{t+1} = w_t - \lambda \frac{\delta L}{\delta w_t} \tag{2.2}$$

where w_t are the parameters at time t, L is some loss function, λ is the step size, which says how big steps the parameters should take towards the calculated gradient. A commonly used gradient descent algorithm is Stochastic Gradient Descent (SGD) [34]. SGD is very similar to gradient descent. The difference is that SGD stochastically chooses one or a subset of training samples to perform gradient descent on. Whereas gradient descent uses all the training samples for each update. This makes SGD considerably faster than gradient descent. However, SGD generally does not move towards the true gradient of the loss function. Thus SGD is computationally faster than gradient descent but requires more rounds to converge.

Backpropagation

Backpropagation is the algorithm that determines how the weights and biases should be updated to minimize the loss function. Backpropagation calculates how the weights and biases should be updated to decrease the loss function the most for a single training example. Backpropagation is performed by computing the partial derivatives of the loss function. The partial derivatives are computed with respect to the weights and biases, $\partial L/\partial w^l_{jk}$ and $\partial L/\partial b^l_j$, for all layers. To compute the partial derivatives, backpropagation first computes the error, δ^l_j, for the j^{th} node in layer l. The error is computed by:

$$\delta^l_j = \frac{\partial L}{\partial z^l_j} \tag{2.3}$$

where z_j^l is the weighted input of node j in layer l. To get the partial derivatives from the error, backpropagation will relate δ_j^l to $\partial L/\partial w_{jk}^l$ and $\partial L/\partial b_j^l$.

To relate δ_j^l to $\partial L/\partial w_{jk}^l$ and $\partial L/\partial b_j^l$, four equations are used [30]. Combining those four functions makes it possible to calculate the gradient of the loss function. The first equation, used for calculating the error in the output layer, is:

$$\delta_j^{l_o} = \frac{\partial L}{\partial a_j^{l_o}} \sigma'(z_j^{l_o}) \tag{2.4}$$

Here, the term $\partial L/\partial a_j^L$ shows how the loss is changing with respect to the output activation of neuron j. The term, $\sigma'(z_j^{l_o})$ shows at which rate the activation function σ changes with the weighted input $z_j^{l_o}$.

The second equation uses the error in the next layer, δ^{l+1} to calculate the error in the current layer, δ^l:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z') \tag{2.5}$$

Thus by combining Equation 2.4 and Equation 2.5, the error for any layer δ^l can be computed by using the error at the output layer and propagating backward through the layers.

The third equation shows how the change of loss is affected in relation to the bias of any node in any layer:

$$\frac{\partial L}{\partial b_j^l} = \delta_j^l \tag{2.6}$$

The partial derivative with respect to the bias, $\partial L/\partial b_j^l$, is exactly equal to the error δ_j^l. Since combining Equation 2.4 and Equation 2.5 results in $\delta_j^l \in \delta^l$, one of the two partial derivatives has been linked to the error.

The forth equation shows how the change of loss is affected in relation to an individual weight in any layer:

$$\frac{\partial L}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \tag{2.7}$$

The term a_k^{l-1} is the activation output of the node in the previous layer. The term δ_j^l is the node error in the current layer. Both terms are known how to compute, thus the second partial derivative has been linked to the error. Combining the four equations results in the final backpropagation method. Gradient descent can then be used together with the backpropagation method to update the weights and biases of the network.

Overfitting

One problem which can occur when training a model is overfitting. When a model is overfitted it has been trained to model the training data too closely. It models noise in the data which doesn't appear in other data. Overfitting makes the model worse in classifying new data. There are several ways to reduce the risk of overfitting.

One method is to decrease the number of features the model is trained on. In this method, you remove unnecessary input that acts as noise. Another method is regularization. Regularization penalizes large model parameters and therefore creates a less complex model. A less complex model has a reduced opportunity to overfit. Another method to reduce the risk of overfitting is dropout. Dropout ignores some nodes in the hidden layers. Ignoring a node means that the node, and all its connected edges, are not used in that training round. More specifically, for each node, there is a probability p of that node being ignored. Dropout helps reduce overfitting by reducing the probability of co-dependency between nodes.

2.2 Federated learning

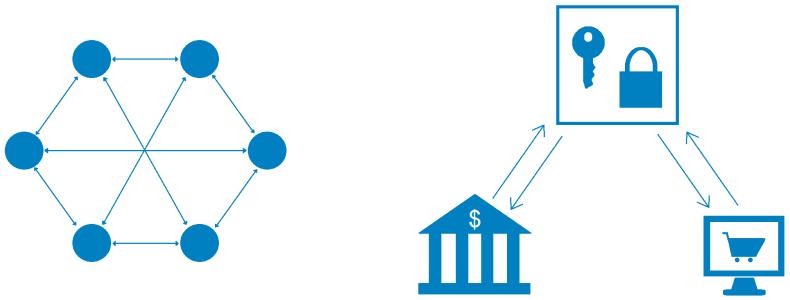
Federated learning is a technique to leverage private data without sending it to a central server. The term was first used by a paper from google [28]. Their goal was to "present a practical method for the federated learning of deep networks based on iterative model averaging". Federated learning is defined to solve problems with the following four properties[28]:

- The data belonging to one participant may not be representative of the data belonging to the other participants
- Some participants will have more training time than others
- The number of participants will be larger than the average amount of data per participant
- The participants are limited in their communication

Federated learning, in its basic form, consists of a central server and loosely connected participants (called clients). The server coordinates the sharing of knowledge between the clients. The sharing can be done through many different types of strategies. However, it is done without sharing the actual data contained on the clients. Instead, the clients send their model parameters to the server which combines them. This combination is then returned to the clients. The clients then continue training on the combined model and again return the results to the server. This is the approach of the original FedAvg strategy [28].

Since federated learning first was introduced, many different approaches have been proposed. A survey on federated learning [29] classified those approaches in 5 ways: network topology, data partition, data availability, aggregation and optimization algorithm, and open-source frameworks.

Network topology refers to how the training and aggregation of models are performed. In the original centralized approach, as previously mentioned, the server simply manages and aggregates the training from the asynchronous updates it receives from the clients. Another approach is user clustering, where users are grouped based on how similar their data distribution is. This clustering is used to reach convergence faster. The third approach is distributed (fully decentralized) federated



(a) A distributed learning topology

(b) Vertical learning

Figure 2.3: Two types of federated learning. 2.3b illustrates a bank and a shop using a coordinator to train together without revealing private information

learning. Here, each participant shares the information in a peer-to-peer fashion, i.e., there is no central server. The distributed topology is illustrated in Fig. 2.3a.

Data partition refers to the way the data is partitioned between different participants. The division can be made broadly into horizontal, vertical, and transfer learning. Horizontal learning is defined as when each participant's dataset shares the same features, but the instances are different. This is the classical federate learning, e.g., when many different phones are used to train text prediction. Vertical federated learning is defined by the use of different feature spaces to jointly train a model. For instance, when a bank and an internet shop try to train a model together. The bank has credit data while the internet shop has order data. This data could be used to jointly train a purchase prediction model. Vertical learning is illustrated in Fig. 2.3b. The last approach is federated transfer learning. Here traditional ML transfer learning techniques (i.e., knowledge from solving one problem is used to solve another) are applied in the federated domain.

Data availability refers to the availability of the participants to train. There are two categories, cross-silo FL and cross-device FL. In cross-silo FL the number of participants is usually in the range of 2-100, the devices are indexed and usually always available. In contrast, the cross-device approach usually has a very large number of participants. The connections are also less reliable and the participation is more random.

Aggregation and optimization strategy controls the training flow. In centralized federated learning, the strategy chooses what clients it will use for training. It then combines the returning parameters, from the clients, at the end of the round. A distributed federated learning strategy manages the interaction of the participants. For instance, assign the participants temporary roles as servers and clients. The first proposed centralized strategy was FedAvg, described in Algorithm 1, in which the server acts as an orchestrator by sending out its parameters to a random subset of clients. These clients train on the parameters and then return them to the server. The server makes an average and repeats the process. Some other variants are:

- SMC-AVG: A strategy based on Secure Multiparty computing. It allows distrustful participants to collaboratively calculate sums of their private values, without revealing their own private values to the other participants. This allows the clients to share knowledge without revealing their gradients. Which makes it even harder to derive private information.
- FedMa: was proposed for creating CNN and LSTM (Long Short-Term Memory) updates in an FL environment.
- FedMatch: a semi-supervised federated algorithm.

Federated learning can be implemented using many different frameworks. Some examples are TensorFlow federated, PySyft, Flower, and FATE. However, most are catered to the centralized topology.

ALGORITHM 1: FedAvg

```
RunServer():
    initialize w_0
    for each round t = 1,2,... do
        m ← max(C · K, 1)
        S_t ← (random sets of m clients)
        for each client k ∈ S_t in parallel do
            w^k_{t+1} ← ClientUpdate(k, w_t)
        end
        w_{t+1} ← Σ_{k=1}^{K} (n_k / n) w^k_{t+1}
    end
RunClient(k, w):
    B ← (split P_k into batches of size mathcalB)
    for each local epoch e from 1 to E do
        for batch b ∈ B do
            w ← w − η∇ℓ(w; b)
        end
    end
return
```

2.3 Semi-supervised learning

Semi-supervised learning (SSL) tries to solve machine learning problems with a small amount of labeled data and a larger amount of unlabeled data. The idea is to leverage the knowledge from the labeled data together with some underlying structure of the unlabeled data. For instance, if we create a text classifier to classify what sort of food a certain recipe describes, e.g., Italian, French, or Chinese. We may have a lot of pizza recipes labeled as Italian. These recipes most often include tomato sauce as an ingredient. Therefore, our classifier will find that tomato sauce is a strong indicator of Italian food. This is where semi-supervised learning can be leveraged. In the unlabeled data, there may be a lot of recipes that use both the words olive oil and tomato sauce. These two words might also often be connected

with pasta. This could guide the classifier toward labeling pasta recipes as Italian. Despite not getting any such labeled examples. A two dimensional classification problem on synthetic data is shown in Fig. 2.4

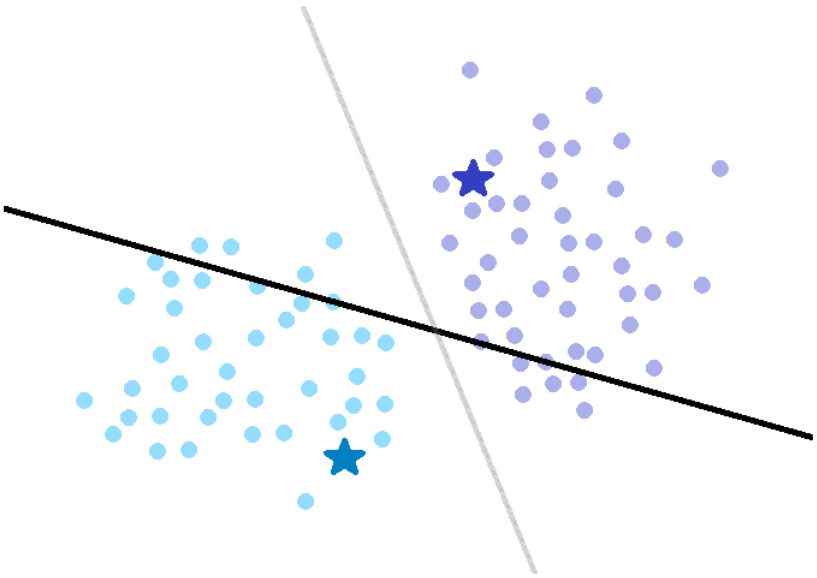


Figure 2.4: A semi-supervised example. The stars represent labeled data. The black line represents the decision boundary for only training on the labeled data. The grey line represents the decision boundary for training on the labeled data and the unlabeled data

A necessary condition for semi-supervised learning to be useful is that the underlying marginal data distribution, $p(x)$, over the input space contains some information about the posterior distribution, $p(y|x)$ [40], i.e, the input data must contain some information about the desired output. This condition has been shown to be true in most cases, by the successful application of semi-supervised learning to many learning problems. However, the interaction between $p(x)$ and $p(y|x)$ is not always the same. These interactions have been formalized using three assumptions: points that are close to each other should share a label, clusters often share a label, and the data lies roughly on a manifold (an object that looks flat to an ant in a lower dimension, e.g., a sphere in 3 dimensions) with a lower dimension than the input space.

According to [40] semi-supervised algorithms can be either inductive or transductive. The goal of inductive methods is to create a classifier that can make predictions for any object in the input space. In contrast, transductive methods do not create a classifier for the entire input space. Rather, its predictive power is limited to the exact data it has experienced during training.

The inductive methods can be further divided into three categories. Firstly, wrapper

methods let the model be trained in a supervised fashion. Then the classifier is used to construct pseudo-labeled data, which is then used to train the model. The second method is unsupervised clustering, which can be done in three ways. Either useful features are extracted, the data is pre-clustered, or the initial parameters for the supervised training are chosen with an unsupervised method. The last method is to directly include unlabeled data into the objective function or optimization procedure of the learning method.

All transductive methods are either explicitly graph-based or can be understood in such a manner. This is caused by the fact that there exists no model of the input space. Instead, information is propagated via connections of the data points. Transductive graph-based methods can be generalized into three steps. First, a set of objects is used to construct a graph where similar points are connected. The second step adds weights, to the edges, to capture how similar the nodes are. In the third step, some chosen method is used to assign labels to the unlabeled data points.

One wrapper method is *consistency regularization*. *Consistency regularization* was first proposed by Bachman et al. [5]. *Consistency regularization* is based on the assumption that a model should output similar predictions when fed slightly augmented inputs. *Consistency regularization* combines supervised training on labeled data and training on unlabeled data using the loss function:

$$\sum_{b=1}^{\mu B} ||P_m(y|\alpha(u_b)) - p_m(y|\alpha(u_b)||_2^2 \tag{2.8}$$

where α and p_m are some stochastic functions and therefore give the two terms different values. This loss function encourages smoothness of the model around the labeled data. An example of how augmentation functions affects synthetic data in a two dimensional classification problem is shown in Fig. 2.5. [12] made some empirical observations motivating why consistency regularization would improve model accuracy. The first observation was that networks trained on noisy data demonstrate a generally lower consistency than those trained on clean data. The second observation was that consistency reduces more significantly around noisy-labeled data points than correctly-labeled ones. The noise in these labels eventually becomes modeled in training, i.e. the model becomes overfitted and loses accuracy. These two observations suggest that maximizing consistency of prediction could improve robustness to noise. [12] also observed a correlation between validation accuracy and training consistency.

Another wrapper method is *pseudo-labeling*. *Psuedo-labeling* uses the model itself to obtain artificial labels for unlabeled data. These pseudo labels can then be used to train the model. However, the pseudo labels are only retained if the confidence of the model is higher than some predefined threshold. The loss function for pseudo-labeling can be defined in the following way:

$$\frac{1}{\mu B} \sum_{b=1}^{\mu B} \mathbb{1}(max(q_b) \geq \tau) H(\hat{q}_b, q_b) \tag{2.9}$$

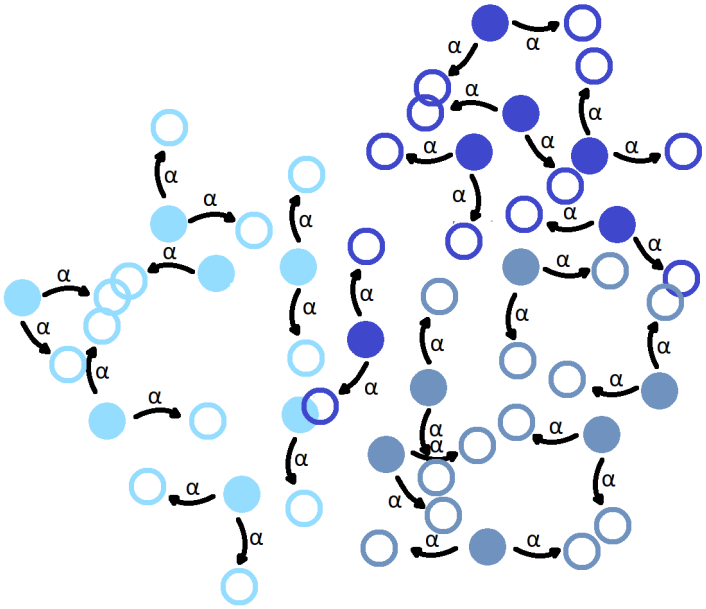


Figure 2.5: An example of consistency loss. The arrows represent data augmentations. Two random augmentations are applied to each data point. After the augmentations, the two new points should likely remain in the same class.

where $q_b = p_m(y|u_b)$, $\hat{q}_b = argmax(q_b)$, τ is the threshold and argmax is applied to produces a "one-hot" probability distribution. Psuedo-labeling can be theoretically motivated by the clustering assumption, i.e., data points of the same class are located near each other. When models are only trained on a few data points, it causes some labeled points to become classified with low confidence of several classes instead of high confidence of one class. Psuedo-labeling uses the clustering assumption to encourage unlabeled data to get one label with high confidence.

2.3.1 FixMatch

FixMatch is a state-of-the-art SSL algorithm [38]. It is based on two SSL approaches *Consistency regularization* and *pseudo-labeling*. FixMatch is described in Algorithm 2

The loss function in Fixmatch consists of two terms, a supervised loss l_s and an unsupervised loss l_u. The supervised loss is a standard cross-entropy loss on weakly augmented input:

$$\ell_s = \frac{1}{B}\sum_{b=1}^{B} H(p_b, p_m(y|\alpha(x_b))) \tag{2.10}$$

The unsupervised part of Fixmatch computes a pseudo-label for each unlabeled datapoint. The pseudo-label is then used in a standard cross-entropy loss function. The pseudo-label is produced by first making a prediction on weakly augmented data $q_b = p_m(y|\alpha(u_b))$. In the case of image classification, this could be a rotation. Then the pseudo-label becomes $\hat{q}_b = argmax(q_b)$. The pseudo-label will then be

used together with a strongly augmented version of u_b in cross-entropy loss:

$$\ell_u = \frac{1}{\mu B} \sum_{b=1}^{\mu B} \mathbb{1}(max(q_b) \geq \tau) H(\hat{q_b}, p_m(y|\mathcal{A}(u_b))) \tag{2.11}$$

where τ is a confidence threshold that denotes when to use a pseudo-label. The full loss will be $\ell_s + \lambda_u \ell_u$.

As strong augmentation functions, $\mathcal{A}$, [38] used RandAugment[10] and CTAugment (Control theory augment)[6]. RandAugment was developed as an augmentation function for image data. RandAugment randomly applies some N transformations sequentially to an image. The transformations used are:

- Rotate, rotates the image some random degrees
- Shear-x, shears the image along the horizontal axis with
- Shear-y, shears the image along the vertical axis
- Sharpness, randomly adjusts the sharpness of the image
- Contrast, adjusts contrast
- Translate y, translates the image horizontally
- Translate x, translates the image vertically
- Solarize, inverts all pixels above a threshold
- Posterize, reduces the number of bits for each color channel
- Color, randomly changes the color
- Brightness, adjusts the brightness
- AutoContrast, maximizes the image contrast by setting the darkest pixels to black and the lightest pixels to white

Each transformation in RaundAugment is linearly scaled with the hyperparameter M. Only using one hyperparameter to scale all transformations was motivated experimentally [10]. Since auto augment only has two hyperparameters (M and N) it can be optimized by grid search.

Unlike RandAugment, CTAugment doesn't need hyperparameter tuning [6]. This makes it easier to use with a small number of labeled examples. CTAugment divides each parameter for each transformation into bins of distortion magnitudes. Let m be a vector of bin weights for some parameter. The bin weights are initialized to 1 and are used to choose which magnitude bin to apply an image. At each training step, two transformations are sampled for each image in a uniformly random manner.

To augment the images, two transformations are sampled and each of their parameters gets a modified set of bin weights, $\hat{m}$, produced. Where $\hat{m}_i = m_i$ if $m_i \geq 0.8$ and $\hat{m} = 0$ otherwise. Magnitudes are then sampled from $Categorical(Normalize(\hat{m}))$.

The weights of the two transformations are updated by sampling each magnitude bin in m_i uniformly randomly. The resulting transformations are applied to a labeled

example x with label p to obtain an augmented version, $\hat{x}$. The correctness of the models prediction is then measured with:

$$1 - \frac{1}{2L} \sum |p_{model}(y|\hat{x}; \theta) - p| \tag{2.12}$$

The weight for each sampled magnitude bin is then updated as:

$$m_i = pm_i + (1 - p) \tag{2.13}$$

where $p = 0.99$ is a fixed exponential decay hyperparameter.

ALGORITHM 2: FixMatch algorithm

Input Labeled batch $\mathcal{X} = \{(x_b, p_b) : b \in (1, ..., B)\}$, unlabeled batch $\mathcal{U} = \{u_b : b \in (1, ..., \mu B)\}$, confidence threshold τ, unlabeled data ratio μ, unlabeled loss weight λ_u
$\ell_s = \frac{1}{B} \sum_{b=1}^{B} H(p_b, \alpha(x_b))$
for $b = 1$ ***to*** μB **do**
  $q_b = p_m(y|\alpha(u_b); \theta)$
end
$\ell_u = \frac{1}{\mu B} \sum_{b=1}^{\mu B} \mathbb{1}(max(q_b) \geq \tau) H(argmax(q_b), p_m(y|\mathcal{A}(u_b)))$
return $\ell_s + \lambda_u \ell_u$

2.4 Distributed algorithms

Distributing algorithms is a technique to enable computation on many independent nodes without the need for a central coordinator. The nodes can range from cores in a multi-core processor to geographically distributed computation nodes, connected in a network. Each node should run a different part of the algorithm, while having limited knowledge about the other nodes.

A network of nodes can be built up by different system architectures. Three different types of systems are: centralized, decentralized, and distributed. As shown by fig 2.6, the difference is how the nodes are connected. In a centralized system, there exists one server to which all other nodes are connected and communication is always done between a single node and the server. This system has one big limitation, it is prone to failure. If the server goes down, the whole system crashes. A decentralized system works similarly. However, in this system, there exists more than one server. The nodes can communicate with any of the servers, and the servers communicate with each other. By having many servers, the network is better protected against failure. In a distributed setting, all nodes are connected to each other. Each node in the distributed network has the same authority. There exists no server to handle coordination. Thus, specific distributed algorithms have to be used to handle coordination in a distributed system.

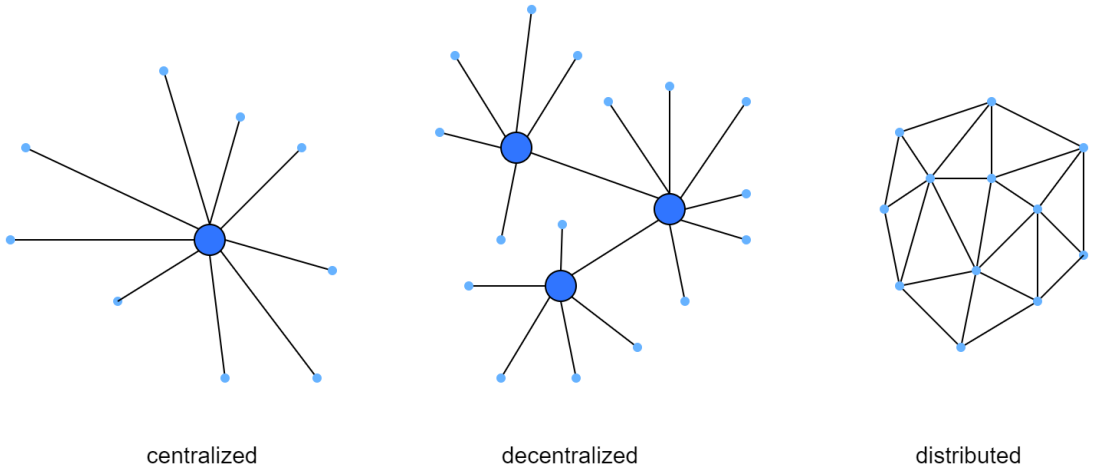


Figure 2.6: A diagram illustrating the difference between a centralized, decentralized, and distributed network topology.

To perform the coordination between nodes in a distributed system, there exist many algorithms. One common approach among distributed algorithms is to choose one node as coordinator [11]. The node to be selected as coordinator generally does not matter. However, the selection of the coordinator has to be performed in agreement with all nodes. Every node in the network should be aware of who is selected as the current coordinator. One such technique of choosing a coordinator is leader election. The challenge leader election tries to address is how to make a uniform decision of coordinator among all the nodes.

Another way to model a distributed algorithm is a transition system. A transition system is used to define the behavior of discrete systems, such as finite-state machines. A finite-state machine usually consists of a graph with nodes and edges, where each node represents a state and each edge represents an action. There are various types of transition systems that can model different types of algorithms. One such system is the Channelled Transition System (CTS) which was created to model the use of communication channels [2]. While CTS make the abilities of a single agent clear, it may not represent the interaction of agents clearly. ReCiPe, was developed to give better semantics for such interactions [2, 1].

2.4.1 Model of Communication

We present a modified version of the ReCiPe communication formalism [2, 1]. We start by specifying agents (or programs) and their local behaviours and we show how to compose these local behaviours to generate a global (or a system) one. We assume that agents rely on a set of data variables D, and a set of multicast channels CH.

Definition 1. *An agent is* $A_i = \langle V_i,\ g_i^r,\ \mathcal{T}_i^s,\ \mathcal{T}_i^r, \theta_i \rangle$*, where:*

- *V_i is a finite set of typed local variables, each ranging over a finite domain. We use V' to denote the primed copy of V (to store the next assignment to variables in V) and Id_i to denote the assertion $\bigwedge_{v \in V_i} v = v'$.*
- *$g_i^r(V_i, \text{CH})$ is a receive guard describing the connection of an agent to channel ch.*
- *$\mathcal{T}_i^s(V_i, V_i', N, \text{D}, \text{CH})$ is an assertion describing the send transition relation, where N specifying the number of required receiving agents. That is, given the current assignment to V, the agent can send a message D to N agents on some channel in CH, and consequently the next assignment to V is stored in V'.*
- *$\mathcal{T}_i^r(V_i, V_i', \text{D}, \text{CH})$ is an assertion describing the receive transition relation. That is, given the current assignment V, an agent can receive message D on some channel in CH and evolve to V'.*
- *θ_i is an assertion on V_i describing the initial states, i.e., a state is initial if it satisfies θ_i.*

Agents exchange messages. A message is defined by the channel it is sent on and the data it carries. A set of agents agreeing on the data variables D, and channels CH define a *system*.

We informally define the transition relation of the system ρ as follows: The transition relation ρ relates a system state s to its successors s' given a message $m = (ch, \mathbf{d}, k)$ by agent k. Namely, there exists an agent k that sends a message with data $\mathbf{d}$ (an assignment to D) on channel ch to n other agents if and only if there exists n other agents are connected and participate in the interaction. That is, the agents are connected to channel ch (i.e., each agent j in state s^j has $g_j^r(s^j, ch)$ satisfied) get the message and perform a receive transition. As a result of interaction, the state variables of the sender and these receivers might be updated.

This means that the interaction is blocking both for the sender and the receivers. That is, a sender is not able to send without having the required number of receivers available.

3

Methods

This chapter will explain the methods used in this project. It will begin to explain how data was gathered for the road surface noise dataset. In the following section, both datasets used in the evaluation will be described. Section three will explain how and what features will be extracted from the audio data. The following section will describe the model architecture of the classifier and the proposed distributed training algorithm, distributed FixMatch. Furthermore, we describe the proposed augmentation function for audio features. The last two sections will describe the simulation environment and the evaluation methods.

3.1 Data Gathering

The data was recorded through several short road trips. The road trips were performed on the road surfaces; asphalt, gravel, oil gravel, cobblestone, and snowy roads. To record the sound of the wheel against each road surface, a microphone was used. The microphone was connected to the vehicle. The placement of the microphone was tested at different positions, both inside and outside the car. The placement chosen was close to the wheel on the outside of the car. To minimize wind noise, the microphone was placed inside the wheel well. The back right wheel well was chosen to also minimize engine noise, as seen in Fig. 3.1.

Every car produces different sounds when driving. This is due to a number of different parameters that varies based on the car. For example; the engine type, the airflow, the tire type, the tire wear, motor defects, etc. Thus when training a neural network on sounds from just one car, the network model could be adapted to the sounds from that specific car. The model will then perform poorly when applied to different cars. To minimize this effect and make the model more general, the data used in training should be as generic as possible. To make the data more

Model	Year	Type	Form	Motor	Tires
Nissan Qashqai	2011	Diesel	Cross over	1.6-dCi	Stud-free winter tires
Volvo v70	2013	Diesel	Station wagon	D4	Stud-free winter tires
Polestar 2	2022	Electric	Hatch back	single motor	Stud-free winter tires

Table 3.1: Parameters of the cars used for recording

generic, the car parameters should change between the recordings. To achieve data generality we collected recordings with several different car models. The car models used for recordings are presented in Table 3.1. There are other factors that can vary apart from just the car. The weather, road quality, and car speed are such things. To minimize the effect of these parameters, recordings were collected on; wet and dry roads, on roads of different quality, and at different car speeds. The car speeds varied from 10 km/h to 80 km/h.

Figure 3.1: The placement of the microphone when collecting road noise for the road surface noise dataset.

3.2 Datasets

Two datasets were used when testing distributed FixMatch, the proposed distributed semi-supervised algorithm. Firstly, the road surface noise dataset, which was recorded during the project. Secondly, UrbanSound8K, which is an open-source dataset used to evaluate audio classifiers.

3.2.1 Road surface noise dataset

The first dataset contains audio recordings of road noise that were recorded during the project. The dataset contains 5 different classes which represent audio from 5 road surfaces:

- 4688 clips of gravel road noise
- 4249 clips of asphalt road noise

- 1130 clips of cobble road noise
- 2010 clips of oil gravel road noise
- 667 clips of snow road noise

Each clip is one second long and there are 12744 in total, which have been cut from about 212 minutes of audio recordings. When listening to the clips we thought it was often clear what type of road surface the sound has been recorded on. However, some of the clips were recorded when driving faster than 50 km/h, which has caused some audio clipping. This could possibly have been caused by wind. These clips were removed and are not included in the dataset.

3.2.2 UrbanSound8K

The second dataset, UrbanSound8K [35], is a dataset of urban sounds. UrbanSound8K was chosen as an additional dataset for two reasons. It has been used in previous research, which makes comparisons in performance easier. It is a harder classification problem than the road surface noise dataset, which could show that distributed FixMatch works on harder problems. The aim of UrbanSound8K was to create a dataset consisting of the most frequently occurring urban noise complaints. The authors had three conditions for the dataset; The sounds should occur in an urban environment, all recordings should be real, i.e. no artificially made sounds, and the dataset should be large enough for training algorithms that can be used on real sensor networks. To achieve this, they utilized the Freesound [17] database. There they searched and downloaded recordings associated with each class. These recordings were then manually checked and cut to only include the relevant sounds. The final dataset consists of 10 classes distributed as follows:

- 1000 clips of street noise
- 1000 clips of child noise
- 1000 clips of dog noise
- 1000 clips of AC noise
- 1000 clips of drill noise
- 1000 clips of jackhammer noise
- 1000 clips of engine noise
- 929 clips of siren noise
- 427 clips of car horn noise
- 374 clips of gunshot noise

In total there were 8732 clips with a maximum length of 4s. The total length of the recordings is 8.5h. However, there were two clips that were shorter than the minimum clip size(60ms) for the feature set used in openSMILE. These clips were removed before training.

3.3 Feature Extraction

We extract features from the audio recordings to feed as input to the neural network. The features are extracted by applying filters to the audio recordings. The filters are applied using the audio feature extraction toolkit, openSMILE [15]. We choose openSMILE for its ease of use and to ensure reproducibility. The filters extract different Low-level descriptors. The low-level descriptors include audio features such as:

- Frame Energy
- Frame Intensity / Loudness
- Mel-spectograms
- Auditory Spectra

The choice of OpenSMILE was based on previous state-of-the-art audio classification papers[3][4]. In [3] the authors tried to detect if the road surface the car was driving on was wet or dry. To detect road surface wetness, they extracted a set of features, Auditory Spectral Features (ASF) [27], described in the next paragraph. Since road surface wetness detection and road surface detection are similar problems we will try their set of features. However, openSMILE also has its own predefined feature sets that were made for speech recognition. The feature set from [3] will be compared to the predefined sets in openSMILE and the best one will be used.

3.3.1 Auditory Spectral Features

The first step in making the feature set from [3] is to compute a short-time Fourier transform (STFT). Unlike [3] we use a 100ms frame size and 100ms step size. This is done to reduce the dimensionality of the features. The STFT outputs auditory spectral features which can then be used by 26 triangular filters to derive the Mel spectrograms $M^{100}(n,m)+1)$. Then the Mel spectrogram will be transformed to log-scale 3.1. This is done to match the human perception of loudness since humans can hear the type of road surface they are driving on.

$$M_{log}^{100}(n,m) = log(M^{100}(n,m) + 1.0) \tag{3.1}$$

The positive first-order differences can then be calculated from each log Mel spectrogram:

$$D^{100}(n,m) = M_{log}^{100}(n,m) - M_{log}^{100}(n-1,m) \tag{3.2}$$

The positive first-order differences are used to emphasize sudden changes in the audio, specifically a note onset. The log Mel spectrogram and the first-order difference will give 52 features. Finally, we will include the frame energy and its derivative to get the full feature vector.

Feature set	ComParE_2016	GeMAPSv01b	eGeMAPSv02	ASF
Accuracy	81.62%	81.11%	84.51%	80.60%

Table 3.2: Accuracy of different feature sets when used to train a neural network to classify the Road surface noise dataset

3.3.2 Feature set evaluation

As mentioned earlier, there exist several premade feature sets in openSMILE. The ones we chose to evaluate are; ComPareE_2016 [37], GeMAPSv01b [13] and eGeMAPSv02 [13]. The difference between these feature sets is the Low-Level Descriptors extracted, as well as the functionals applied to the LLD.

The largest of these premade feature sets is ComParE_2016. It extracts 65 LLD and applies various functionals to these LLD to obtain 6373 features. The set was created as a tool to solve four challenges within speech recognition; detect non-linguistic events such as a sigh or laughter, recognize conflict in group discussion, recognize the emotion of a speaker, and determine the pathology of a speaker.

The smallest feature set we evaluated was GeMAPSv01b [13]. To obtain this feature set, various functionals were applied to 18 extracted LLD, resulting in a feature set of 62 features. This set was also created to solve speech recognition tasks. However, the aim of the GeMAPSv01b set was to create a minimalistic set compared to the ComPareE_2016, which could perform comparably in emotion detection. GeMAPSv01b performed similarly, and even better at certain emotions [13], compared to the ComPareE_2016.

The final feature set we evaluated was the eGeMAPSv02 [13]. This feature set is an addition to GeMAPSv01b. It uses the same 62 features as a base. In addition to these features, they additionally extract 7 LLD and apply functionals to these LLD to obtain the final feature set of 88 features.

The different feature sets were evaluated empirically to determine which set would work best with our classification problem. The road surface noise dataset was used to evaluate the feature sets. One by one, each feature set was used to extract features from the dataset. These features were then used as inputs to a central supervised model. The model was trained with the same hyperparameters for all the feature sets. For each feature set, the model was trained for 800 training epochs. The training was performed for ten runs for each set and the highest achieved accuracy among these 10 runs was collected. The highest accuracy for every feature set is shown in Table 3.2. Here we can see that all premade openSMILE feature sets achieved higher accuracy than the feature set from [3]. Among the premade sets, eGeMAPSv02 performed best. Thus we used eGeMAPSv02 as our feature extraction set.

3.3.3 eGeMAPSv02

eGeMAPSv02 [14] consists of 88 features in total. The first step in the extraction of those features is to extract the 25 LLDs. All of these LLDs are smoothed over 3 frames (for pitch, jitter, and shimmer, the smoothing is only performed within regions where fundamental frequency $F_0 > 0$). The 25 LLD can be sorted into three groups: Frequency related parameters, Energy/Amplitude related parameters, and spectral parameters:
Frequency related parameters:

- **Pitch**, logarithmic F0 on a semitone frequency scale, starting at 27.5 Hz (semitone 0).
- **Jitter**, deviations in individual consecutive F0 period lengths.
- **Formant 1, 2, and 3 frequency**, Formant 1, 2, and 3 frequency, centre frequency of first, second, and third formant
- **Formant 2–3 bandwidth**, added for completeness of Formant 1–3 parameters.

Energy/Amplitude related parameters

- **Shimmer**, difference of the peak amplitudes of consecutive F0 periods.
- **Loudness**, estimate of perceived signal intensity from an auditory spectrum..
- **Harmonics-to-Noise Ratio (HNR)**, relation of energy in harmonic components to energy in noiselike components.

Spectral parameters:

- **Alpha Ratio**, ratio of the summed energy from 50–1000 Hz and 1–5 kHz
- **Hammarberg Index**, ratio of the strongest energy peak in the 0–2 kHz region to the strongest peak in the 2–5 kHz region.
- **Spectral Slope 0–500 Hz and 500–1500 Hz**, linear regression slope of the logarithmic power spectrum within the two given bands
- **Formant 1, 2, and 3 relative energy**, as well as the ratio of the energy of the spectral harmonic peak at the first, second, third formant's center frequency to the energy of the spectral peak at F0.
- **Harmonic difference H1–H2**, ratio of energy of the first F0 harmonic (H1) to the energy of the second F0 harmonic (H2).
- **Harmonic difference H1–A3**, ratio of energy of the first F0 harmonic (H1) to the energy of the highest harmonic in the third formant range (A3).
- **Formant 1**, bandwidth of first formant.
- **MFCC 1–4** Mel-Frequency Cepstral Coefficients 1–4.
- **Spectral flux** difference of the spectra of two consecutive frames.

The second step is to use the LLD to compute the 88 features. Firstly, all 25 LLD have the **arithmetic mean** and the **coefficient of variation** functionals applied to them. This results in 50 features. Furthermore, 6 temporal features are added:

- The **rate of loudness peaks**, the number of loudness peaks per second.
- The **mean length** and the **standard deviation** of regions where $F_0 > 0$.
- The **mean length** and the **standard deviation** of regions where $F0 = 0$; approximating pauses).
- The number of continuous regions, where $F_0 > 0$, per second (pseudo syllable rate).

which results in 56 features so far. Another 26 features are applied when $F0$ is non-zero. 16 of those features are added by applying the following functionals to *pitch* and *loudness*:

- **The 20-th,50-th and 80-th percentile**
- **The range of the 20-th to 80-th percentile**
- **The mean and standard deviation of the slope of rising/falling signal parts**

The remaining 10 are computed by taking the arithmetic mean and coefficient of variation of the spectral flux and MFCC 1–4. This results in 82 features in total. 5 additional features are added by applying the following functionals when $F0 = 0$:

- **The arithmetic mean of the Alpha Ratio**
- **The Hammarberg Index**
- **the spectral slopes from 0–500 Hz and 500–1500 Hz**
- **The arithmetic mean of the spectral flux**

Giving us 87 features. Lastly, the **equivalent sound level** is added resulting in 88 features.

3.4 Model Architecture & Training

The features are fed to a fully connected neural network (FNN). The FNN was chosen instead of a CNN since the feature space was small. The FNN consists of on input layer of 88 nodes, three hidden layers of 200 nodes each, and an output layer of 5 or 10 nodes depending on the classification task. The model was implemented in PyTorch. PyTorch was chosen because of its ease of use, its flexibility given by its use of dynamic computational graphs, and since it's often used in research. The training on each node is performed by stochastic gradient descent (SGD) with the loss function described in Algorithm 2. Both labeled and unlabeled data were used. Furthermore, cosine learning rate decay was used. Lastly, a dropout of 50% was used on the first layer during training.

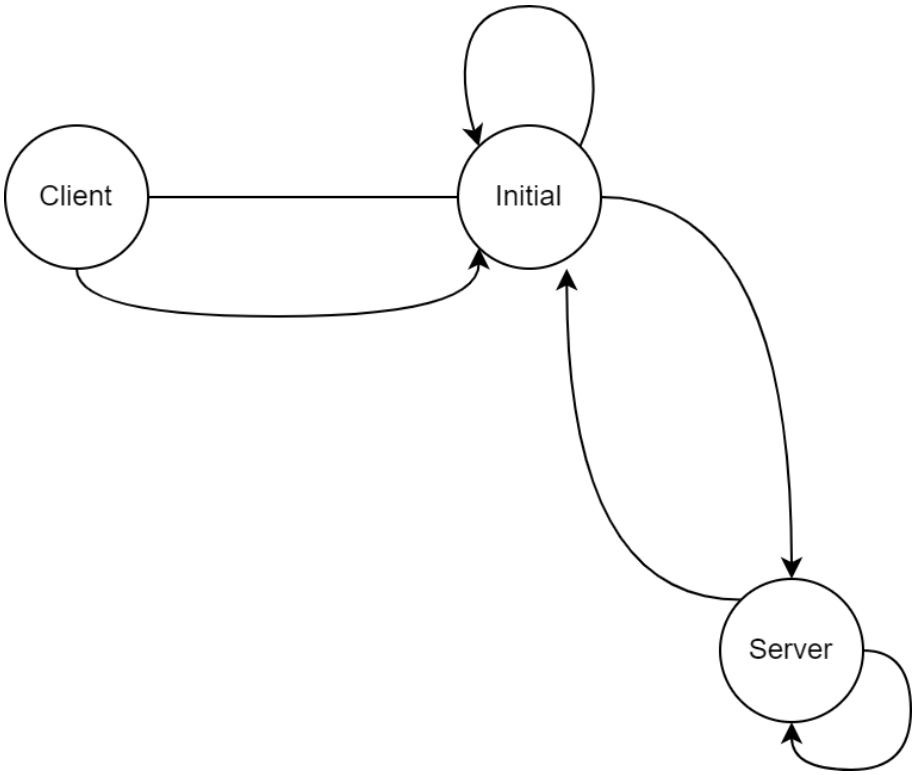


Figure 3.2: State transitions of distributed FixMatch

3.4.1 Distributed FixMatch

Distributed FixMatch combines ideas from FixMatch and FedAvg and distributes them. A key difference between distributed FixMatch and FedAvg, is that FedAvg has a server that performs the coordination of the supervised training on clients. Whereas in distributed FixMatch, every node takes turns performing the server role. The semi-supervised training is inspired by FixMatch but the augmentation functions have been modified to work with audio features extracted by openSMILE.

Each agent A_i in distributed FixMatch has its own set of local variables, that is parameters w_i, its own set of unlabeled data $\mathcal{U}_i$ and an identical copy of labeled data $\mathcal{L}$. The training in distributed FixMatch is performed in rounds R. Each client begins in an initial state 0, where they attempt to select some N of their neighbors to begin a training round. If a agent A_i finds N neighbors in the initial state then it makes a state transition and sends its parameters to the other models. At the same time the N neighbors make a state transition and begin training with the average of agent A_i parameters and their own parameters. The agent A_i acts as a server in this instance. When the neighbors have finished training they return the new parameters to A_i and return to the initial state. A_i now combines the parameters it received from each of its neighbors and averages them. A_i returns to the initial state. The distributed FixMatch algorithm is presented in Algorithm 3. A visualization of the state transitions is shown in Fig. 3.2. The state transitions are performed according to the modified version of ReCiPe introduced in the Model of Communication section in the theory chapter.

ALGORITHM 3: Distributed FixMatch

SelectN is a function, which queries N neighbour clients if they are in the idle state and returns them if true. R is the number of server rounds, w is the model parameters, $\mathcal{U}_i$ is unlabeled data, and $\mathcal{L}$ is labeled data.

Initialize w; $State \leftarrow initial$
Function `Step()`:
  if $State = initial$ **then**
    $nodes \leftarrow SelectN()$
    if $|nodes| = N$ *and* $r \leq R$ **then**
      for $node \in nodes$ **do**
        Transition *node* to the *client* state and send w to *node*
      end
      $State \leftarrow server$
      $r \leftarrow r + 1$
    end
  end
  if $State = server$ **then**
    $\hat{w} \leftarrow$ Get received parameters from clients
    if $|\hat{w}| = N$ **then**
      $w_t \leftarrow \frac{1}{N} \sum_{n=1}^{N} \hat{w}_n$
      $State \leftarrow initial$
    end
  end
  if $State = client$ **then**
    $w_s \leftarrow$ Get received parameters from server
    $w \leftarrow \frac{w+w_s}{2}$
    for each local epoch e from 1 to E **do**
      for batch $b \in (\mathcal{U}_i, \mathcal{L})$ **do**
        $w \leftarrow w - \eta \nabla \ell(w; b)$
      end
    end
    send w to server
    $State \leftarrow initial$
  end

The training is performed by a slightly modified version of FixMatch. In [38] FixMatch is implemented to train an image detection model. The loss function contains one strong augmentation and one weak augmentation function. The strong augmentation function uses RandAugment[10] which randomly chooses between a set of image transformations. These transformations can be flipping the image, changing its colors, or rotating it. The weak augmentation function only uses a subset of these transformations, such as flipping the image. The features we use to classify the audio are 88 scalar features extracted from the eGeMAPSv02 feature set by openSMILE. Therefore, the augmentation functions used in [38] are incompatible with classifying audio features. The scalar features need to be augmented by a different method. One such feature is illustrated in Fig. 3.3. In the image we can clearly see that some classes tend to cluster around a specific value on the feature. The idea is that modifying these features is analogues to image transformations. Changing the loudness or frequency of the audio is similar to rotating or changing the color of an image. Audio clips where many features are similar but slightly different should belong to the same class. We will use this clustering assumption to create our augmentation functions. If we add a small random value from a Gaussian distribution to each feature, then the augmented features should likely remain in the same class:

$$\alpha(u) = u + x \tag{3.3}$$

where $x = \{x_0, x_1, ..., x_{87}\}$, $x_i \sim \mathcal{N}(\mu, \sigma^2)$ and $u \in \mathcal{U}$ is some unlabeled data. The augmentation function α can easily be extended into a strong augmentation function and a weak augmentation function by using different σ values.

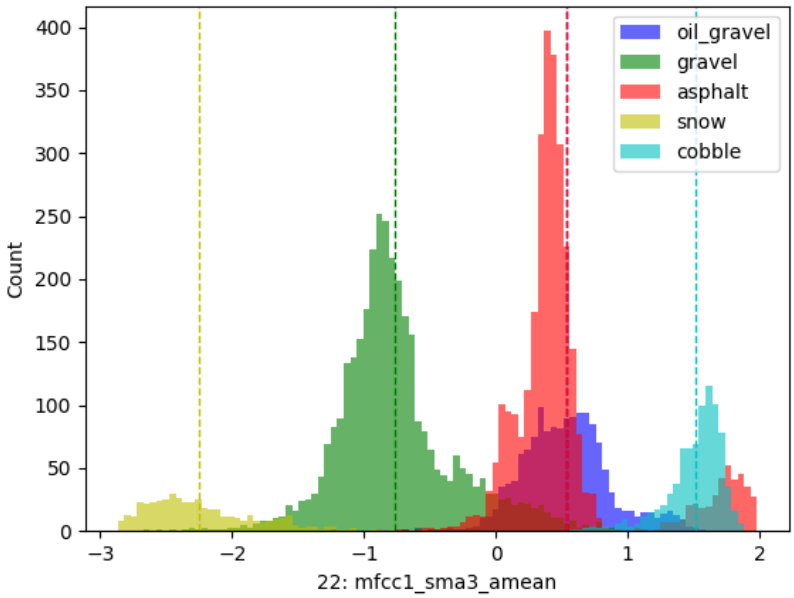


Figure 3.3: The distribution of feature mfcc1_sma3_amean for all road surface data points visualized for each class.

3.5 Hyperparameter optimization

Distributed FixMatch uses several hyperparameters. To achieve the best result we decided to search for the optimal values. This optimization was done using grid search since it is simple to implement. The search was made for three hyperparameters: soft augment σ, strong augment σ, μ, and the threshold τ. For each of these hyperparameters, some interval was chosen. Then a combination of hyperparameters for each point in this interval was made (144 combinations in total). Each combination was then run for 400 iterations. The five best combinations of hyperparameters in terms of accuracy are presented in Table 3.3.

acc	aug_w	aug_s	threshold	mu
66.72	0.2	0.6	0.95	3
66.72	0.4	0.5	0.95	3
66.55	0.3	0.6	0.95	3
66.38	0.2	0.5	0.95	3
66.2	0.1	0.6	0.99	3

Table 3.3: Accuracy of a distributed FixMatch when trained on the UrbanSound8K dataset for different combinations of hyperparameters.

3.6 Simulation

To examine how the algorithm will perform in practice, a simulation environment was implemented. The simulation consisted of a simulation handler and nodes. Each node was implemented as a class with its own model and data sets. Furthermore, each node has a step function that attempts a state transition. The step function is shown in Algorithm 3. The choice of which client gets to run its step function is made in a uniformly random manner by the simulation handler. The simulation is implemented in this way so that it can simulate the parallelism of the nodes in real-world applications. In a parallel system, it is unknown in what order the nodes try to make a state transition. The random transitions give of our simulation the same property.

To distribute the data between the nodes, the datasets were split up into labeled and unlabeled data. An equal share of data from each class was chosen randomly to be used as labeled data. The labeled data were split up into a validation set and a training set. The split was made such that all classes were equally represented in the training set. A copy of the validation and training set was distributed to all the nodes. The remaining data were used as unlabeled data. This data was spread out to the nodes in an IID distribution. I.e., each node was given the same amounts of unlabeled data per class and all data on each node were distinct. [21].

3.7 Evaluation

To evaluate the performance of the distributed FixMatch algorithm we will perform several tests. Firstly, distributed FixMatch will be compared to a supervised centrally trained model. The supervised model is trained using stochastic gradient descent with cross-entropy loss. The supervised model and the semi-supervised distributed FixMatch will both have access to the same set of labeled data. However, each participating node in distributed FixMatch will also have access to its own set of unlabeled data. One test will be performed with the UrbanSound8K dataset and one test will be performed with the road surface sound dataset. The training will then be performed five times for 800 rounds to maximize the probability that the model achieved its highest accuracy. From these 5 runs, the highest accuracy was collected. The two methods will then be compared by their accuracy in the classifications. These tests are performed to evaluate the performance of distributed FixMatch compared to only using labeled data with a central model. The tests will also be performed with several different amounts of labeled data. This is done to evaluate how the amount of labeled data affects the performance of distributed FixMatch with respect to prediction accuracy. The results of these tests are shown in Fig. 4.1 and in Fig. 4.2. The results show that distributed FixMatch can, in some settings, utilize the unlabeled data to improve the accuracy.

Another test that will be done is to evaluate how the accuracy of distributed FixMatch is affected when increasing the number of clients. This is done to see how well the algorithm scales. These tests will be performed with 5, 10, and 25 clients. The number of clients is however somewhat limited since our dataset is too small to make tests with a realistic amount of clients. The evaluation results of these tests are presented in Table 4.1.

We will also conduct tests to evaluate the communication efficiency of the distributed FixMatch algorithm. These tests will be performed by counting how many communication messages are sent by each node in the network. The number of messages sent will be compared when using the distributed FixMatch algorithm vs when using the federated learning algorithm. To measure the messages, a counter will be increased whenever a message is sent between two nodes. Each node will have an individual counter so the distribution of messages can be seen. The tests will compare the total messages sent in the network before reaching a certain accuracy, as well as the maximum number of messages sent by one node. This accuracy threshold was set to 59% accuracy. The tests will be performed with different number of clients to evaluate the scalability of the distributed FixMatch algorithm. These tests are presented in Table 4.3. They show that the communication overhead of distributed FixMatch is more evenly distributed than in federated learning. They also show that distributed FixMatch scales better than federated learning

4

Results

To evaluate our distributed FixMatch algorithm several tests were performed on the road surface dataset and UrbanSound8K. In the first set of tests, the performance of the semi-supervised component of distributed FixMatch is tested. A model trained by the distributed FixMatch is compared to a model trained using supervised learning. Both models train on the same labeled data but distributed FixMatch also trains on unlabeled data. In these tests, we can see that distributed FixMatch successfully leverages the unlabeled data. For instance, with 50 labels per class on UrbanSound8K, distributed FixMatch achieved an accuracy of 68.51% compared to the supervised training, which achieved 63.12%. The second set of tests evaluates how the size of the network impacts the performance of distributed FixMatch. Each test uses the same data and model. However, the number of clients used varies. These tests show that the network size has little impact on performance, as seen in Table 4.1. Lastly, we demonstrate the individual accuracies of the clients during training.

For the road surface noise dataset, the model trained using distributed FixMatch outperforms the supervised model as seen in Fig. 4.1. This is most prevalent in the scenarios with fewer labels. The accuracy with one label for every class is 85.04% for distributed FixMatch and 68.35% for the supervised. This may seem like a very high accuracy for so few labels. However, this is because the features of the different classes in the road surface noise dataset are quite distinct, e.g. the feature seen in Fig. 3.3. There we can see a clear distinction among most classes. With only one label per class, the supervised model has some problems in classifying the data. A possible explanation of these results is that the supervised model has difficulties learning the real distribution with just one example. This will be easier for distributed FixMatch since its unlabeled data will tend to cluster around the labeled data points. Therefore, it can derive which points likely belong to which distribution, giving it a better idea of the distributions of the features. Nevertheless, the distributed FixMatch model's and the supervised model's accuracies converge with an increased number of labels per class. At 50 labels per class they are the same. This is not surprising since 50 labels per class give the supervised model a good idea of the real distributions of the features. Hence, the advantage distributed FixMatch got from its unlabeled data is strongly diminished.

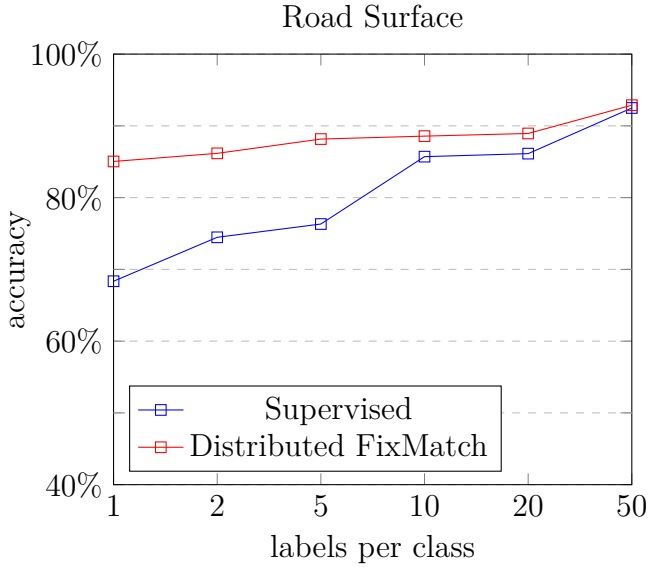


Figure 4.1: The accuracy difference of distributed FixMatch vs a central supervised model for the Road Surface dataset using a fixed number of labeled training examples for each class.

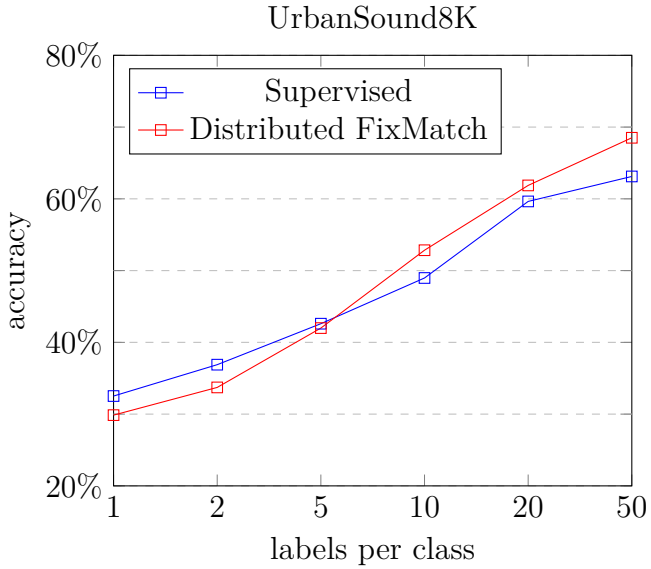


Figure 4.2: The accuracy difference of distributed FixMatch vs a central supervised model for the UrbanSound8K dataset using a fixed number of labeled training examples for each class.

For the UrbanSound8K dataset, we can see that the trend (Fig. 4.2) is different from the previous test. Here, the distributed FixMatch model performs slightly worse than the supervised model on the tests with few labels. This could possibly be because UrbanSound8K is harder to classify than the road surface noise dataset. The feature distributions in UrbanSound8K are less clustered, e.g. shown in Fig. 4.3, this could cause distributed FixMatch to miss-label some of the unlabeled data.

For instance, the model could receive labeled data with an outlier feature. This could cause distributed FixMatch to wrongly classify data from other categories in the same category. This will cause the model to be trained on incorrect labels. In contrast, the supervised model could be more conservative in assuming that clusters near labeled data belong to the same class. When the amount of data increases the risk of only receiving outliers decreases and the chance of getting labeled data points near the real class clusters increases. Therefore, as the number of labels increases the performance difference between the supervised and distributed FixMatch is reversed. With greater than 5 labels distributed FixMatch outperforms the supervised model. Finally, with 50 labels per class distributed FixMatch outperforms the supervised model by 5.39 percentage points.

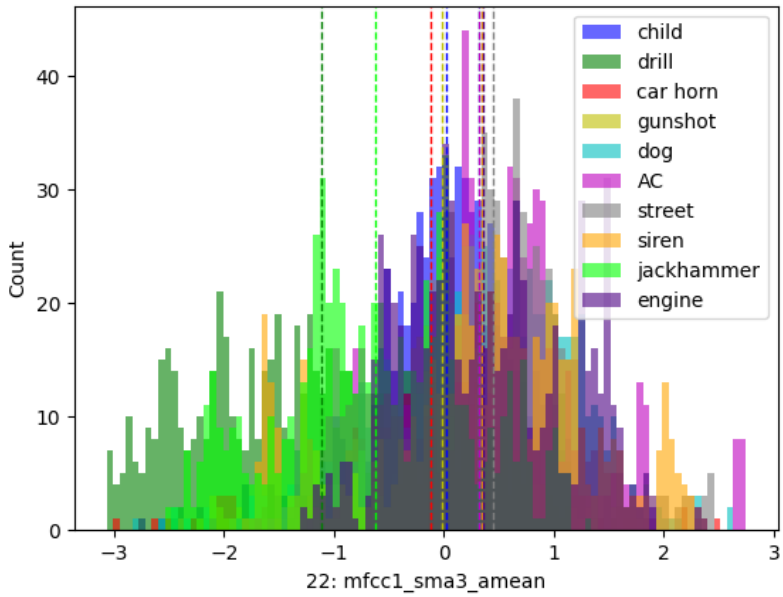


Figure 4.3: The distribution of feature mfcc1_sma3_amean for all UrbanSound8K data points visualized for each class.

In Table 4.1 the test results when training with different number of clients are shown. For both datasets the difference in accuracies is small. The accuracies for UrbanSound8K differ by 2.26 percentage points. The accuracies for the road surface noise dataset differ by 2.24 percentage points. The accuracy for UrbanSound8K slowly decreases with the number of clients. Suggesting a scalability problem. However, the road surface noise dataset first increases and then decreases with the number of clients. This suggests that the variation could be caused by noise. Therefore, the algorithm seems to scale, but more testing could be needed to confirm this.

Fig. 4.4 shows the average accuracy of each client when training on the UrbanSound8K dataset. It also shows the individual accuracy of each client. The accuracy is recorded at each round of the simulation, i.e., each time a client tries to make a state transition. As the rounds progress, the average accuracy slowly increases. In contrast, the increase in accuracy of the individual clients is much more unstable,

Dataset	acc 5 clients	acc 10 clients	acc 25 clients
UrbanSound8K 30 labels	66.72%	65.85%	64.46%
road surface noise dataset 5 labels	88.72%	90.56%	88.32%

Table 4.1: Number of clients affect on accuracy

even if all of them eventually stabilize. This can be explained partly by high accuracy increases during training and partly by the parameter exchanges between the clients. A trained client who exchanges parameters with an untrained client will lose some accuracy due to the nature of the aggregation. This explanation is validated by the convergence of all individual accuracies with the average accuracy during the last rounds of training.

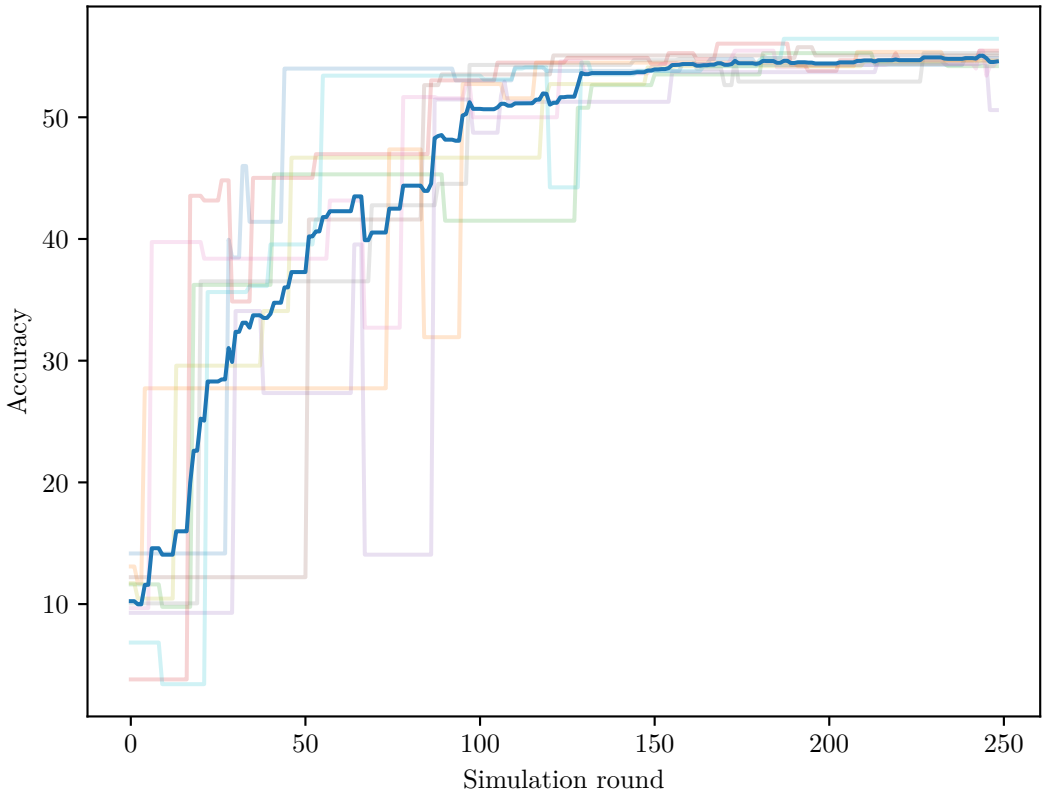


Figure 4.4: Accuracy development of the nodes during training on the UrbanSound8K dataset.

In Fig. 4.5 we can see how well a model trained using distributed FixMatch could classify the different road surfaces. We can see that the model can classify snow and cobblestone with nearly 100% accuracy. This outcome is somewhat predictable. This is because the noise coming from snow and cobblestone is easier to distinguish compared to the other road surfaces. Oil-gravel and gravel were predicted with fairly

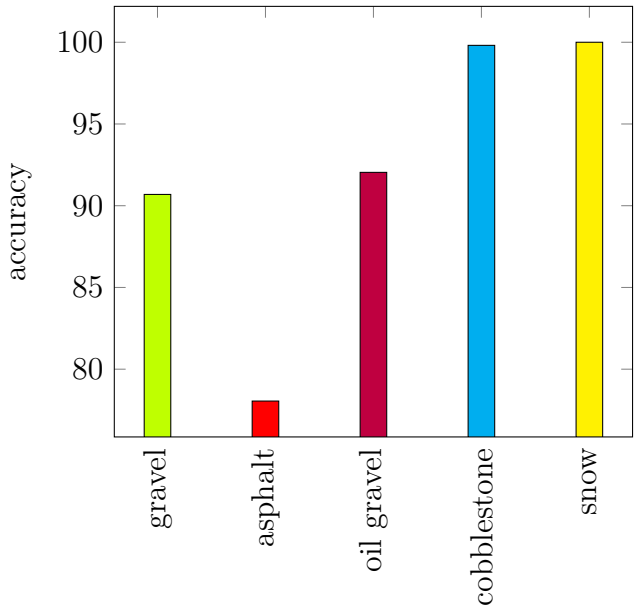


Figure 4.5: Accuracy of the different road surfaces. Trained using distributed FixMatch. Road surface noise accuracy was taken when a client reached at least 87% accuracy. An average of 5 runs was taken

high accuracy. The model had the most difficulty classifying noise coming from an asphalt road. Here the model only classified it correctly about 78% of the time. The lower accuracy could have been caused by the more diverse dataset. For example, unlike the other classes, the asphalt recordings were collected by all the different vehicles used. The asphalt recordings were also recorded on many different roads with different road quality, making the data more scattered. Another possible cause is that asphalt does not have an equally distinct sound compared to the other road surfaces.

4.1 Communication overhead

In this section, we will first make a theoretical comparison of communication overhead between centralized training (FixMatch using a central dataset), federated training (FedAvg combined with FixMatch using a distributed dataset), and distributed FixMatch (using a distributed dataset). Then we will empirically demonstrate the communication overhead of the different methods.

In the theoretical comparison, we will first demonstrate the complexity of the total communication overhead, i.e., all communication that takes place on the network. Secondly, we will demonstrate the complexity of the maximum local cost, i.e. the highest cost of communication for any single node in the network. The results of both comparisons are shown in Table 4.2.

To train on all the data with a centralized approach, it would be required to upload the audio data from each of the participants. This would result in a communication

	Total cost	Maximum node cost
Centralised	$O(n \cdot d)$	$O(n \cdot d)$
Federated	$O(n \cdot r \cdot m)$	$O(n \cdot r \cdot m)$
distributed FixMatch	$O(n \cdot r \cdot m \cdot degree(network))$	$O(degree(network) \cdot r \cdot m)$

Table 4.2: Communication complexity. Where n is the number of nodes participating in the training, d is the client dataset size, r is the number of server rounds during training and m is the model size.

overhead of $O(n \cdot b)$ where n is the number of participants and b is the average dataset size. On the contrary, the total communication overhead of federated training and distributed FixMatch would be $O(n \cdot r \cdot m)$ where n is the number of participants, r is the number of communication rounds and m is the model size. In the federated algorithm, the server sends to and receives its model m from some fraction of the n clients every round, until round r, i.e a communication of $O(n \cdot r \cdot m)$. In the distributed FixMatch algorithm, every node can act as a server for r rounds. Each round it acts as a server it sends and receives its model m from $degree(network)$ clients. Thus the communication of each node is $O(r \cdot m \cdot degree(network))$ and the communication of the entire network is $O(r \cdot m \cdot degree(network) \cdot n)$. Comparing $O(n \cdot b)$ and $O(r \cdot m \cdot n)$, b tends to be larger than $m \cdot r$. For instance, one second of a recorded wav file takes 172 KB. The average daily drive time is 51 minutes [23]. Therefore, the average amount of data would be $51 \cdot 60 \cdot 172$ KB = 526MB daily. This can be compared to the model size used in this project of 0.4 MB. The average amount of rounds needed for convergence on the UrbanSound8K(6.6 GB) dataset is 24.24, giving a total communication overhead per client of $24.24 \cdot 0.4\text{MB} \approx 9.7\text{MB}$.

Another way to evaluate the communication overhead is to look at the maximum communication overhead of each participant. This is where distributed FixMatch has a large advantage over the other methods. In the centralized approach, the server communicates with n clients. From each client it receives some b amount of data. This gives it a communication overhead of $O(n \cdot b)$. The client only communicates with the server, by uploading the data b, which gives a communication overhead of $O(b)$. In the federated approach, the maximum local communication overhead also occurs on the server. The server receives communications with a fraction of n clients every round. Therefore receiving $O(m \cdot n)$ data every round, where m is the model size. This gives a communication overhead of $O(n \cdot r \cdot m)$. The clients only communicates with the server, by receiving and sending the model m during r rounds, which gives a communication overhead of $O(n \cdot r)$. In distributed FixMatch, the maximum communication overhead a node will receive is when it acts as a server. The temporary server will send and receive m from each of the temporary clients during r rounds. The number of temporary clients is limited to the number of neighbors a client has, i.e. degree(network). This gives a communication overhead of $O(degree(network) \cdot r \cdot m)$. In the worst case, a node acting as client can communicate with all of it's neighbours all of their rounds r. In this case, it sends and receives the model m to each of the neighbours r times. This give a worst communication overhead of $O(degree(network) \cdot r \cdot m)$, which is less than or equal to

the communication overhead when acting as a server. To summarize, the maximum local communication overhead increases linearly, with the number of nodes in the network, for the federated and centralized approach. But for distributed FixMatch, the communication overhead only increases linearly for its neighbors.

To measure communication overhead, we performed a test that evaluated the number of communication messages sent before convergence. We tested on the UrbanSound8K dataset using federated training (FedAvg combined with FixMatch using a distributed dataset) and distributed FixMatch. To measure the number of communications needed for both algorithms, we set a cutoff accuracy that determined when the algorithm had converged. For federated training, the accuracy was examined on the server. For distributed FixMatch, the accuracy was examined on every node since every node can act as a server. The results of the tests are presented in Table 4.3.

In communication tests there are several things to note. For distributed FixMatch, the average node communicates a relatively similar amount as the most strained node in the network. when using 10 nodes the average node sent $242.4/10 \approx 24.2$ messages, while the most strained node sent 32.6 messages. This is also true when increasing the number of nodes in the network. On the contrary, for federated training, the most strained node is not close to the average communication overhead. The average overhead with 10 nodes is $158.0/10 = 15.8$, compared to the server which has a communication overhead of 80. Additionally, the communication overhead of the most strained node for distributed FixMatch is fairly constant independently of the network size, as seen in Fig. 4.6. The maximum number of communications is connected to the degree of the network, which does not change when the total number of nodes increases. The total communication cost of distributed FixMatch is almost the same for 25 and 50 nodes. This is not inline with the theoretical prediction. This might be because in the theoretical analysis the algorithm stops after a certain number of rounds, while in these tests it stops by an achieved accuracy. The federated algorithm has a linear increase in both total communication costs and the maximum local cost with respect to the number of nodes.

Algorithm	Total communications	Most strained
Distributed FixMatch 10 nodes	242.4	32.6
Distributed FixMatch 25 nodes	396.4	25.6
Distributed FixMatch 50 nodes	387.8	15.0
Federated training 10 nodes	158.0	80.0
Federated training 25 nodes	607.0	306.0
Federated training 50 nodes	1670.0	840.0

Table 4.3: Number of communication messages sent for distributed FixMatch vs federated training until achieving a threshold accuracy

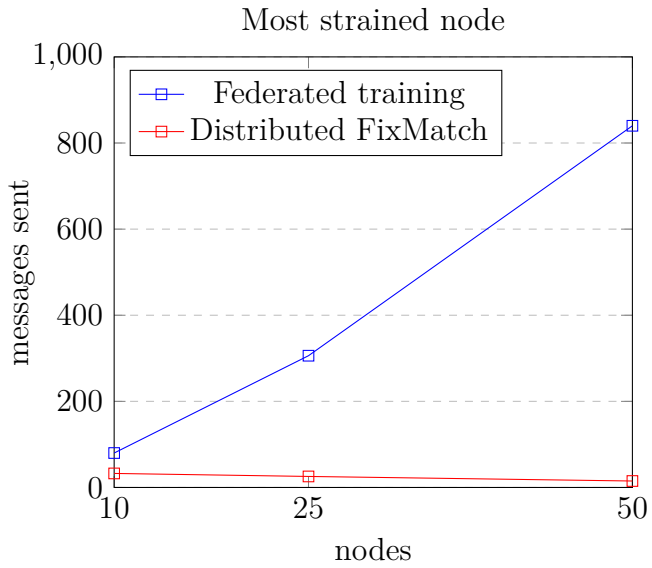


Figure 4.6: The communication overhead of the most strained node for Distributed FixMatch vs Federated training

5

Concluding remarks

In this chapter, we will first discuss the results from the previous chapter, especially relative to the goals and challenges stated in the introduction chapter. After that, we will state the conclusion we think can be drawn from the results and the project in its entirety. Lastly, we will speculate about improvements that could be made to the method and what further research could be of interest.

5.1 Discussion

As shown in the results chapter, FixMatch outperformed the supervised approach in the audio classification tasks. Furthermore, distributed FixMatch performs similarly to other state-of-the-art semi-supervised approaches. For instance, the performance of distributed FixMatch was 68.5% for 50 (500 total) labels per class on the UrbanSound8K dataset. While, semi-supervised learning on the same dataset, where a maximum of 75.17% $\pm$ 1.52 was achieved [26] when using 600 total labels. These two results indicate that distributed FixMatch was successful in leveraging unlabeled distributed data on many different clients. Furthermore, the accuracies of distributed FixMatch were achieved without sending any raw data, thus preserving the privacy of participants. The comparable accuracy also suggests that the openSMILE feature set, eGeMAPSv02, is possible to use for more problems than its original use case of speech recognition. These results also imply that the simple augmentation of features with a Gaussian distribution could be a useful technique for pseudo labeling. However, the accuracy advantage over the centralized method varied with different datasets and amounts of labeled data. Therefore, the pseudo-labeling method may be more or less useful in different situations.

The theoretical and empirical analysis of the communication costs of distributed FixMatch shows that it scales better than its centralized alternatives. The theoretical analysis states that the maximum node cost should increase with $O(degree(network) \cdot r \cdot m)$, where r is the number of rounds and m is the model size. This theoretical cost is in line with testing, Fig. 4.6. The maximum local communication overhead does not increase with the network size. Furthermore, we can see in Table 4.1 that the size of the network has little impact on the accuracy of the models. In comparison, the federated algorithm (FedAvg combined with FixMatch) has a theoretical maximum local communication overhead of $O(n \cdot r \cdot m)$, where n is the number of nodes,

r is the number of rounds and m is the model size. This theoretical analysis is also confirmed by tests (Fig. 4.6). The centralized algorithm's (FixMatch) maximum node communication overhead also increases linearly, with the number of clients $O(n \cdot d)$. However, it depends on the data size instead of the model size. Therefore, which approach is optimal could depend on other factors than the number of nodes, e.g., model size and data size. However, in practice, the data size tends to be at least as big as the model size. The advantage of the distributed approach is in line with previous theoretical results of similar distributed algorithms [25].

The performance of distributed FixMatch was evaluated using the accuracy metric. Other performance metrics such as precision, recall, F1-score, etc. were not used. One potential problem with only evaluating accuracy is that accuracy does not account for imbalanced class distributions. However, as seen in Fig. 4.5, this is not a problem in our study. Furthermore, distributed FixMatch was tested against a supervised training algorithm instead of a state-of-the-art semi-supervised algorithm. This could make comparisons between distributed FixMatch and the alternatives more difficult.

The road surface noise dataset has some limitations. For example, only a few cars were used in its recording. Furthermore, the dataset was fairly small with only 3.5 hours of recordings. Additionally, some classes were only recorded with one car. This probably causes a bias in the classifier and it is unclear how well it would generalize. Still, the classifier was able to classify road surfaces of audio recordings from the different roads with high accuracy, as shown in Fig. 4.1. This suggests that road surface detection through audio recognition is possible in real applications.

Another aspect which was not explored in this project is security risks. A vehicle with malicious intent could send corrupt data, for example bad model parameters, to the other vehicles and cause them to lose knowledge. By doing this the bad model parameters sent by the malicious vehicle could spread through the network and cause many vehicles to have corrupted classifiers.

5.2 Conclusion

The results of this project suggest that distributed federated learning can be successfully used in an automotive environment, which requires a minimized network overhead, semi-supervision, and privacy preservation. The implemented algorithm successfully achieved the design goals stated in the introduction chapter. Firstly, distributing the training of the models increased the privacy of the participants. No raw data was uploaded from the participantLastlys. Furthermore, the proposed algorithm, distributed FixMatch, was shown to have lower communication costs than alternative solutions, e.g. a FedAvg and FixMatch combination. , a model trained with distributed FixMatch was shown to achieve better accuracies than a supervised model when they receive the same amount of labeled data. Additionally, distributed FixMatch had comparable accuracies to semi-supervised algorithms on the same dataset [26].

5.3 Future work

Distributed FixMatch was only tested in a limited simulation environment. To further evaluate how the algorithm performs, a more realistic environment could be set up. This could range from parallel execution and communication to deploying it in vehicles. The simple augmentation technique of the features using a Gaussian distribution proved to be working during this study. However, the original FixMatch algorithm performs many different deliberate augmentations. These augmentations are a big part of FixMatch. So a potential improvement could be to further study how features in the audio environment could be augmented while still belonging to the same class.

Other future work to be done is to collect more road surface noise data. By doing this, the algorithm could be evaluated in a more realistic scenario where the data is collected by many different cars. Thus the generality of the algorithm could be analyzed, as well as the scalability. Furthermore, future work could look in to the security risks with the proposed algorithm.

Bibliography

[1] Yehia Abd Alrahman, Giuseppe Perelli, and Nir Piterman. Reconfigurable interaction for MAS modelling. In Amal El Fallah Seghrouchni, Gita Sukthankar, Bo An, and Neil Yorke-Smith, editors, *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems, AAMAS '20, Auckland, New Zealand, May 9-13, 2020*, pages 7–15. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[2] Yehia Abd Alrahman and Nir Piterman. Modelling and verification of reconfigurable multi-agent systems. *Auton. Agents Multi Agent Syst.*, 35(2):47, 2021.

[3] Irman Abdić, Lex Fridman, Daniel E Brown, William Angell, Bryan Reimer, Erik Marchi, and Björn Schuller. Detecting road surface wetness from audio: A deep learning approach. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 3458–3463. IEEE, 2016.

[4] Livio Ambrosini, Leonardo Gabrielli, Fabio Vesperini, Stefano Squartini, and Luca Cattani. Deep neural networks for road surface roughness classification from acoustic signals. In *Audio Engineering Society Convention 144*. Audio Engineering Society, 2018.

[5] Philip Bachman, Ouais Alsharif, and Doina Precup. Learning with pseudo-ensembles. *Advances in neural information processing systems*, 27, 2014.

[6] David Berthelot, Nicholas Carlini, Ekin D Cubuk, Alex Kurakin, Kihyuk Sohn, Han Zhang, and Colin Raffel. Remixmatch: Semi-supervised learning with distribution alignment and augmentation anchoring. *arXiv preprint arXiv:1911.09785*, 2019.

[7] David Berthelot, Nicholas Carlini, Ian Goodfellow, Nicolas Papernot, Avital Oliver, and Colin A Raffel. Mixmatch: A holistic approach to semi-supervised learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[8] Pew Research Center. Americans and privacy: Concerned, confused and feeling lack of control over their personal information. 2019.

[9] Lei Chen. *Curse of Dimensionality*, pages 545–546. Springer US, Boston, MA, 2009.

[10] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[11] MohammadReza EffatParvar, Nasser Yazdani, Mehdi EffatParvar, Aresh Dadlani, and Ahmad Khonsari. Improved algorithms for leader election in distributed systems. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 2, pages V2–6. IEEE, 2010.

[12] Erik Englesson and Hossein Azizpour. Consistency regularization can improve robustness to label noise. *arXiv preprint arXiv:2110.01242*, 2021.

[13] Florian Eyben, Klaus R Scherer, Björn W Schuller, Johan Sundberg, Elisabeth André, Carlos Busso, Laurence Y Devillers, Julien Epps, Petri Laukka, Shrikanth S Narayanan, et al. The geneva minimalistic acoustic parameter set (gemaps) for voice research and affective computing. *IEEE transactions on affective computing*, 7(2):190–202, 2015.

[14] Florian Eyben, Klaus R. Scherer, Björn W. Schuller, Johan Sundberg, Elisabeth André, Carlos Busso, Laurence Y. Devillers, Julien Epps, Petri Laukka, Shrikanth S. Narayanan, and Khiet P. Truong. The geneva minimalistic acoustic parameter set (gemaps) for voice research and affective computing. *IEEE Transactions on Affective Computing*, 7(2):190–202, 2016.

[15] Florian Eyben, Martin Wöllmer, and Björn Schuller. Opensmile: the munich versatile and fast open-source audio feature extractor. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1459–1462, 2010.

[16] Roberto Fierimonte, Simone Scardapane, Aurelio Uncini, and Massimo Panella. Fully decentralized semi-supervised learning via privacy-preserving matrix completion. *IEEE transactions on neural networks and learning systems*, 28(11):2699–2711, 2016.

[17] Frederic Font, Gerard Roma, and Xavier Serra. Freesound technical demo. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 411–412, 2013.

[18] FutureBridge. Artificial intelligence reshaping the automotive industry, April 2020. Accessed on: Dec. 1, 2021. [Online]. Available: `https://www.futurebridge.com/industry/perspectives-mobility/artificial-intelligence-reshaping-the-automotive-industry/`.

[19] Leonardo Gabrielli, Livio Ambrosini, Fabio Vesperini, Valeria Bruschi, Stefano Squartini, and Luca Cattani. Processing acoustic data with siamese neural networks for enhanced road roughness classification. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7, 2019.

[20] Alon Halevy, Peter Norvig, and Nanediri Fernando. The unreasonable effectiveness of data. *Intelligent Systems, IEEE*, 24:8 – 12, 05 2009.

[21] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[22] Wonyong Jeong, Jaehong Yoon, Eunho Yang, and Sung Ju Hwang. Federated semi-supervised learning with inter-client consistency & disjoint learning. *arXiv preprint arXiv:2006.12097*, 2020.

[23] Tefft B.C. Kim W., Añorve V. American driving survey, 2014 – 2017 (research brief). *Washington, D.C.: AAA Foundation for Traffic Safety*, 2019.

[24] Brian Kinniry. More auto computers means more complicated, costly and longer repairs, August 2016. Accessed on: Dec. 2, 2021. [Online]. Available: `https://ceinetwork.com/news-archive/auto-computers-means-complicated-costly-longer-repairs/`.

[25] Xiangru Lian, Ce Zhang, Huan Zhang, Cho-Jui Hsieh, Wei Zhang, and Ji Liu. Can decentralized algorithms outperform centralized algorithms? a case study for decentralized parallel stochastic gradient descent. *Advances in Neural Information Processing Systems*, 30, 2017.

[26] Kangkang Lu, Chuan-Sheng Foo, Kah Kuan Teh, Huy Dat Tran, and Vijay Ramaseshan Chandrasekhar. Semi-supervised audio classification with consistency-based regularization. In *INTERSPEECH*, pages 3654–3658, 2019.

[27] Erik Marchi, Giacomo Ferroni, Florian Eyben, Leonardo Gabrielli, Stefano Squartini, and Björn Schuller. Multi-resolution linear prediction based features for audio onset detection with bidirectional lstm neural networks. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 2164–2168. IEEE, 2014.

[28] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[29] Viraaji Mothukuri, Reza M. Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[30] Michael A Nielsen. *Neural networks and deep learning*, volume 25. Determination press San Francisco, CA, USA, 2015.

[31] David Norrman, Josef Haddad, and Victor Sellstedt. Fixmatch on audio data.

[32] Chunjong Park, Chulhong Min, Sourav Bhattacharya, and Fahim Kawsar. Augmenting conversational agents with ambient acoustic contexts. In *22nd International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–9, 2020.

[33] Josep M Pujol, Vijay Erramilli, Georgos Siganos, Xiaoyuan Yang, Nikos Laoutaris, Parminder Chhabra, and Pablo Rodriguez. The little engine (s) that could: scaling online social networks. *ACM SIGCOMM Computer Communication Review*, 40(4):375–386, 2010.

[34] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[35] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A dataset and taxonomy for urban sound research. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 1041–1044, 2014.

[36] Simone Scardapane, Roberto Fierimonte, Paolo Di Lorenzo, Massimo Panella, and Aurelio Uncini. Distributed semi-supervised support vector machines. *Neural Networks*, 80:43–52, 2016.

[37] Björn Schuller, Stefan Steidl, Anton Batliner, Julia Hirschberg, Judee K Burgoon, Alice Baird, Aaron Elkins, Yue Zhang, Eduardo Coutinho, Keelan Evanini, et al. The interspeech 2016 computational paralinguistics challenge: Deception, sincerity & native language. In *17TH Annual Conference of the International Speech Communication Association (Interspeech 2016), Vols 1-5*, pages 2001–2005, 2016.

[38] Kihyuk Sohn, David Berthelot, Nicholas Carlini, Zizhao Zhang, Han Zhang, Colin A Raffel, Ekin Dogus Cubuk, Alexey Kurakin, and Chun-Liang Li. Fixmatch: Simplifying semi-supervised learning with consistency and confidence. *Advances in Neural Information Processing Systems*, 33:596–608, 2020.

[39] Vasileios Tsouvalas, Aaqib Saeed, and Tanir Ozcelebi. Federated self-training for semi-supervised audio recognition, 2021.

[40] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440, 2020.

[41] Richard D De Veaux and Lyle H Ungar. Multicollinearity: A tale of two nonparametric regressions. In *Selecting models from data*, pages 393–402. Springer, 1994.

[42] Voxforge.org. Free speech... recognition (linux, windows and mac) - voxforge.org. `http://www.voxforge.org/`. accessed 06/25/2014.

[43] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[44] Qizhe Xie, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. *Advances in Neural Information Processing Systems*, 33:6256–6268, 2020.

[45] Enjian Yao, Zhiqiang Yang, Yuanyuan Song, and Ting Zuo. Comparison of electric vehicle's energy consumption factors for different road types. *Discrete Dynamics in Nature and Society*, 2013.

[46] Valentina Zantedeschi, Aurélien Bellet, and Marc Tommasi. Fully decentralized joint learning of personalized models and collaboration graphs. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 864–874. PMLR, 26–28 Aug 2020.

[47] Valentina Zantedeschi, Aurélien Bellet, and Marc Tommasi. Fully decentralized joint learning of personalized models and collaboration graphs. In *International Conference on Artificial Intelligence and Statistics*, pages 864–874. PMLR, 2020.

A

Appendix 1

Dataset	acc 1 label	acc 2 label	acc 5 label	acc 10 label	acc 20 label	acc 50 label
UrbanSound8K, super	32.52%	36.89%	42.60%	48.97%	59.64%	63.12%
UrbanSound8K, semi	29.86%	33.72%	41.99%	52.84%	61.87%	68.51%
Road surface, semi	85.04%	86.17%	88.16%	88.57%	88.94%	92.88%
Road surface, super	68.35%	74.47%	76.32%	85.71%	86.13%	92.49%

Table A.1: Accuracies depicted in Figure 4.2 and Figure 4.1