Reproducing a large-scale Speaker Verification System

Sydney Nguyen
Project Thesis 1
CAI, ZHAW School of Engineering
Winterthur, Switzerland
Technikumstrasse 71

***Abstract*—In recent years, the field of machine learning has experienced tremendous growth, resulting in the development of complex and large-scale systems. However, reproducing the results of these systems has become a major challenge. This thesis presents a comprehensive examination of the challenges and solutions involved in reproducing the results of a speaker verification system as described in the paper ”In defence of metric learning for speaker recognition”. The study focuses on reproducing the baseline speaker verification system presented in the paper, which is selected based on its level of detail and consideration of training time and computational resource constraints. Through the application of standards and guidelines, this study identifies gaps and improves reproducibility. The results show that an equal error rate (EER) of 2.5% is achieved, which is close to the EER value of 2.22 reported in the original paper. The difference in results is attributed to the limited training time used in this study. Despite these limitations, this thesis demonstrates the successful reproduction of the baseline result, and sets the base for further research and advancements in the field of speaker verification. The proposed methods can be utilized to develop more robust and accurate speaker verification systems in the future. This study highlights the importance of documenting intermediate results during the training process in order to have a better understanding of the learning process and make informed assumptions for future improvements.**

***Index Terms*—speaker recognition, speaker verification, reproducibility, equal error rate, machine learning, standards**

I. Introduction

Speaker verification is a crucial task in the field of machine learning, as it has the potential to revolutionize various areas such as security, forensics, and personalization. Speaker verification is a technique used to identify the speaker of a given speech signal by comparing it to a previously enrolled speaker model.

With the rise of big data, the use of deep learning models in speaker verification has grown rapidly, as they are able to effectively handle the large amount and complexity of unstructured data. However, with the increasing complexity of these models, it has become increasingly important to ensure their reproducibility.

Reproducibility is fundamental requirement for the advancement of knowledge and scientific progress. It is important to differentiate between the concepts of reproducibility and replication. A study is considered reproducible if, by using the same data and analysis code, an independent group can obtain the same results as the original study. However, reproducibility does not guarantee that the study's findings are accurate, it only means that the results can be verified by another researcher not involved in the original study. A study is considered replicable if another independent group researching the same phenomenon reaches the same conclusion after conducting similar experiments or analysis on new data. [2] Reproducibility is the ability to obtain consistent computational results using the same input data, computational steps, methods, code, and conditions of analysis. In speaker verification, reproducibility is particularly important as it ensures the reliability and robustness of the model. To accomplish this, researchers should have access to detailed information such as data splitting procedures, model architectures, and hyperparameter values. Additionally, data, models, and code should be made publicly accessible for other scientists to reproduce the work. This allows for reproducibility and the ability to detect any potential biases or bugs. Without this level of transparency, it becomes difficult for other scientists to reproduce the work and identify any issues post-hoc. In order to ensure the reliability of machine learning models, reproducibility should be given high priority.

This study aims to reproduce a large-scale speaker verification system described in the paper ”In defence of metric learning for speaker recognition” [10], referred to as the baseline system. The process of this study involves the following steps:

1) Analysis of the baseline system and selection of the setup based on implementation details and computational limitations.
2) Application of standards and guidelines to identify gaps and improve reproducibility.
3) Implementation of the baseline speaker verification system.
4) Reproduction of the training process and evaluation of performance.
5) Comparison of the results to the ones reported in the original paper.
6) Discussion of the findings and conclusion.

II. Related Work

This chapter will review the existing literature on reproducibility in machine learning and speaker recognition.

A. Reproducibility

1) Reproducibility Challenges for Machine Learning: Reproducibility in machine learning refers to the ability to consistently achieve similar results when running an algorithm on specific datasets [13]. Reproducibility in machine learning can be challenged by a number of factors. These include:

- A lack of records, which makes it difficult to replicate results when inputs and decisions are not recorded.
- Changes in data, which can make it impossible to obtain the same results when the data used in the original work is altered.
- Inconsistency in hyperparameters, which can yield different results when default values are changed during experimentation and not properly recorded.
- Randomization, which is prevalent in machine learning and can affect reproducibility.
- The experimental nature of machine learning, which can make it hard to keep track of important details as changes are made to algorithms, data, and environments.
- Changes in machine learning frameworks and libraries, which can cause different results when a version used to achieve a certain outcome is no longer available or when switching from one framework to another.
- GPU floating-point discrepancy, which can also lead to different results due to hardware, software, or compiler settings.
- Nondeterministic algorithms, which can generate different outputs for the same input at different runs, posing a greater reproducibility challenge.

2) Reproducibility Standards: An attempt has been made to make research reproducible in a paper titled "Reproducibility standards for machine learning in the life sciences" [3]. It suggests standards that involve the publication of data, models, and code, programming best practices and the utilization of workflow automation. The paper suggests that it is challenging to establish a one-size-fits-all standard for the reproducibility of machine learning systems. As an alternative, the authors propose three standards with different levels for reproducibility. The Bronze standard involves making the data, models, and code accessible to the public. This is the minimum requirement for reproducibility as without these elements, reproduction is not possible. The Silver standard adds to the Bronze standard by ensuring that the dependencies can be installed, providing documentation on how to reproduce the work, including details such as the operating system and resource requirements. This standard offers a balance between minimal availability and full automation. Works that comply to this standard will require less time to reproduce than those only meeting the Bronze standard. The Gold standard builds upon the Silver standard by making the analysis reproducible with a single command, resulting in full automation. Works that meet this standard is claimed to be easily reproducible. Apart from the Bronze-Silver-Gold standards, the authors also debate the issue of compute-intensive analysis. Analysis can be demanding in terms of computational resources and can take a lot of time to complete. In some instances, the time it takes to run the analysis may be so long that it becomes almost infeasible for another research group to reproduce it. To address this issue, authors should track and share intermediate outputs, making it possible for others to verify the final results even if they cannot run the entire pipeline themselves.

B. Automatic Speaker Recognition

1) Speech Representation: Spectrogram: Speech can be represented in a number of ways for speaker verification tasks, however one of the most common ways is through the use of spectrograms. A spectrogram is a visual representation of the frequency components of a speech signal over time. It is obtained by applying a Fourier transform to the speech signal, which separates the signal into its different frequency components. The speech signal is divided into small overlapping segments and the Fourier transform is applied to each segment. The resulting frequency components are then represented in a 2D plot where the x-axis represents time and the y-axis represents frequency. The amplitude of each frequency component is represented by a color, with brighter colors indicating higher amplitudes. [8]

Mel Spectrograms are a variant of the spectrogram. They are generated in a way to mimic the way the human auditory system responds to different frequencies and results in a representation that is more representative of the speech signal as perceived by a human. [8] Mel Spectrograms are typically used as input features to machine learning models in speaker verification tasks [10], where they are used to extract discriminative information from the speech signal that can be used to distinguish between different speakers.

2) Speaker Recognition: Speaker recognition tasks involve identifying and verifying the identity of a speaker based on their voice. Some common speaker recognition tasks include [5]:

- Speaker Identification: This task involves determining the identity of a speaker from a given speech sample. It involves training a model on a dataset of labeled speech samples from multiple speakers and then using the trained model to predict the identity of a speaker from a new, unseen speech sample.
- Speaker Verification: This task involves determining whether a given speech sample belongs to a specific speaker or not. This involves comparing a new speech sample to a set of known speech samples from a specific speaker, and determining whether the new sample belongs to the same speaker or not.
- Speaker Diarization: This task involves segmenting a speech signal into different speakers and then labeling the segments with the corresponding speaker identities.

3) State-of-the-Art Speaker Verification Systems: Speaker verification systems identify and authenticate individuals based on their unique voice patterns. These systems are used to verify the identity of a speaker by comparing an input voice sample to a reference voiceprint stored in a database. They can be used in a variety of applications, such as security systems, voice assistants, and call centers [1].

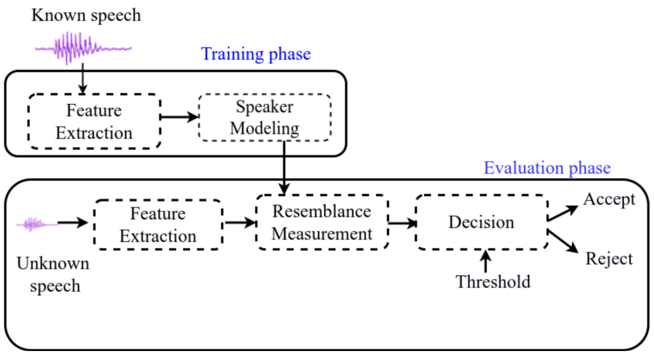


Fig. 1. Block Diagram Speaker Verification System

Figure 1 shows a block diagram of a basic speaker verification system. During the training phase, audio inputs from various speakers are collected and processed. First, the raw audio inputs undergo feature extraction to generate embeddings with their respective labels. These embeddings are then fed into a speaker modeling stage. When an unknown speech sample is received by the speaker verification system, it is compared to the samples that the model is familiar with. A resemblence measure is performed and a decision is made based on a threshold value, either to accept or reject the speaker verification.

There are several state-of-the-art speaker verification systems that are widely used in the research community and industry, some of which include:

- i-Vector based systems: i-Vector based systems are widely used for speaker verification tasks. These systems use a technique called factor analysis to extract a low-dimensional speaker representation called an i-vector from a speech signal. These i-vectors are then used to train a classifier such as a support vector machine (SVM) or a probabilistic linear discriminant analysis (PLDA) to perform speaker verification. [14]
- Deep Neural Network (DNN) based systems: DNNs have been shown to be very effective in speaker verification tasks. These systems use a deep neural network architecture, such as a convolutional neural network (CNN) or a recurrent neural network (RNN), to extract features from a speech signal. These features are then used to train a classifier to perform speaker verification. [9]
- x-vector systems: x-vector systems are an extension of DNN-based systems, which are widely used for speaker verification. These systems are based on a deep neural network architecture called an x-vector, which is trained to extract speaker-discriminative features from a speech signal [15].
- Joint Factor Analysis (JFA) based systems: JFA is a speaker verification technique that uses two factor analysis models to extract speaker-discriminative information from speech signals. The first model extracts the speaker-specific information, and the second model extracts the channel-specific information [12].
- End-to-end systems: End-to-end systems are a recent trend in speaker verification, which use a single neural network to extract features and perform speaker verification in one step. These systems have been shown to be effective in speaker verification tasks, as they are able to learn the best feature representation and classifier in a single network [7].
- Adversarial training based systems: The Adversarial Speaker Verification (ASV) method involves the use of a deep embedding that is trained to be invariant to different conditions through adversarial multi-task training. This involves the joint optimization of two networks: a speaker classification network and a condition identification network. The aim is to minimize the speaker classification loss while simultaneously maximizing the condition loss through a mini-maximization process [11].

4) Performance Metrics for Speaker Verification Systems: The evaluation of speaker verification systems can be performed using the equal error rate (EER) and the minimum decision cost function (minDCF). These metrics showcase different aspects of the system's performance and their accuracy is dependent on the number of trials used for calculation. A detection error trade-off (DET) plot can also be utilized to visualize the system's performance. The EER is reached when the false acceptance rate and false rejection rate are equal, and a lower EER indicates improved performance as it represents a decrease in the combined error of false acceptance and false rejection. The decision cost function (DCF) considers both the cost of each error and the prior probability of target and impostor trials. The DCF can be calculated as

$$DCF = C_{miss} \cdot P_{miss} \cdot P_{target} + C_{fa} \cdot P_{fa} \cdot P_{impostor} \quad (1)$$

where C_{miss} and C_{fa} are the cost functions for missed detection and false alarm, respectively, and P_{miss} and P_{fa} are the percentage of missed target and falsely accepted impostor trials. P_{target} and $P_{impostor}$ represent the prior probabilities of target and impostor trials. The minDCF is obtained by selecting the minimum value of the DCF, which can be estimated by changing the threshold value.

$$minDCF = min[C_{miss} \cdot P_{miss} \cdot P_{target} + C_{fa} \cdot P_{fa} \cdot P_{impostor}] \quad (2)$$

The minDCF provides a way to evaluate speaker verification by considering the minimum total cost of missed detections and false alarms. [1]

5) In defence of metric learning for speaker recognition: The authors of [10] argue the benefits of using metric learning in speaker recognition. To support their argument, they perform a comprehensive evaluation of various loss functions on the VoxCeleb dataset.

The training data set VoxCeleb2 contains 5994 distinct speakers, also referred to as classes. For classification learning objectives, a mini-batch consists of N utterances, each from a different speaker, and is represented by embeddings x_i and the corresponding speaker label y_i, where the index i ranges from 1 to N and the label y ranges from 1 to C.

The following loss functions are used in their experiments:

- Softmax: The softmax loss uses a softmax function followed by a multi-class cross-entropy loss:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{\exp(W_{y_i}\cdot x_i + b_{y_i})}{\sum_{j=1}^{C}\exp(W_j\cdot x_i + b_j)} \tag{3}$$

  where N is the number of samples in a mini-batch, C is the number of classes, W_j is the weight of the j-th class in the last layer of the trunk architecture, b_j is the bias of the j-th class in the last layer of the trunk architecture x_i is the embedding of the i-th sample in the mini-batch y_i is the label of the i-th sample in the mini-batch. It only addresses classification error and doesn't specifically ensure compactness within the same class or separation between different classes.
- AM-Softmax (CosFace): The softmax loss can be expressed in a new form by normalizing the weights and input vectors so that the resulting probability only depends on the cosine angle between the weights and input vectors. It is also called Normalised Softmax Loss (NSL). The resulting loss function is expressed as:

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s(\cos(\theta_{y_i,i})-m)}}{e^{s(\cos(\theta_{y_i,i})-m)}+\sum_{j=1,j\neq y_i}^{C}e^{s\cos(\theta_{j,i})}} \tag{4}$$

  where N is the number of samples in a mini-batch, C is the number of classes, y_i is the label for sample x_i, $\cos(\theta_{j,i}) = \frac{W_j\cdot x_i}{|W_j||x_i|}$ is the cosine similarity between the normalised weight vector W_j and normalised input vector x_i, and m is the cosine margin. However, the NSL-generated embeddings lack enough discrimination as the NSL only focuses on penalizing classification errors. To address this issue, an angle-based margin m is added to the equation, known as the cosine margin. s is a fixed scale factor to make sure the gradient does not get too small during training.
- AAM-Softmax (ArcFace): ArcFace is similar to CosFace, with the exception that it has an additional angular margin penalty of m added between the x_i and W_yi vectors. This angular margin penalty is equivalent to a geodesic distance margin penalty in a normalized hypersphere.

$$L = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{e^{s(\cos(\theta_{y_i,i})+m)}}{e^{s(\cos(\theta_{y_i,i})+m)}+\sum_{j=1,j\neq y_i}^{C}e^{s\cos(\theta_{j,i})}} \tag{5}$$

For metric learning objectives, a mini-batch contains M utterances from N speakers. Embeddings are $x_{j,i}$, where the index j ranges from 1 to N and the label i ranges from 1 to M.

- Prototypical: Each mini-batch consists of a support set S and a query set Q. The query is the M-th utterance from every speaker. Each query instance is then classified against N speakers during training time. This is done with a softmax over distances to each speaker:

$$L = -\frac{1}{N}\sum_{j=1}^{N}\log\frac{e^{S_{j,j}}}{\sum_{k=1}^{N}e^{S_{j,k}}} \tag{6}$$

  where $S_{j,j}$ is the squared Euclidean distance between query and prototype of the same speaker from the support set.
- Angular Prototypical: The same batch formation as prototypical loss is used. One utterance from every class is reserved as the query. A cosine-based similarity metric is then used with learnable scale and bias:

$$S_{j,k} = w\cdot cos(x_{j,M}, c_k) + b \tag{7}$$

  The angular loss function introduces scale invariance, which improves the robustness of objective against feature variance and leads to more stable convergence [10]. The objective is the same as the prototypical loss, in equation (6)

The authors of [10] conduct numerous experiments, keeping other training elements constant. The results of their experiments suggest that the GE2E and prototypical networks exhibit better performance compared to traditional classification-based methods. Furthermore, the networks trained with vanilla triplet loss showed similar performance to most networks trained with AM-Softmax and AAM-Softmax, but those trained with their proposed angular objective surpassed all comparable methods. The authors also introduce an angular version of the prototypical networks that outperforms the other training methods.

III. Solution and Discussion

This section presents the results of the implementation of the speaker verification system, and provides an evaluation of the success of reproducing the baseline system. The baseline system is implemented according to the methods outlined in section IV, and its performance is evaluated using the equal error rate (EER) and minimum decision cost function (MinDCF) metrics on the VoxCeleb1 dataset. The scores of the evaluation are recorded in a score.txt file to ensure reproducibility of the results.

Figur 2 shows the accuracy in %, the loss and learning rate during the training process. The evaluation of these metrics is conducted on the training set VoxCeleb2 over 160 epochs. These metrics are analyzed at each epoch, as they are crucial for the model's learning. The accuracy curve is showing a strict increase towards 1, leading to the assumption that if the training is continued for a longer period of time, the accuracy would eventually reach 1 for the training set. This is logical because the model will have learned all of the utterances in the training set, allowing it to correctly perform speaker verification for speakers it already knows. The performance of the model is evaluated by measuring the EER and minDCF on the VoxCeleb1 test set during a 160 epoch. These values are plotted in figur 3. The evaluation is carried out every

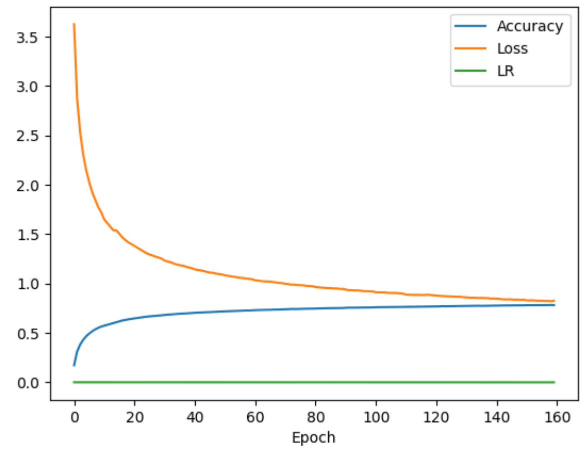


Fig. 2. Accuracy (%), Loss and Learning Rate on VoxCeleb2 Training Set

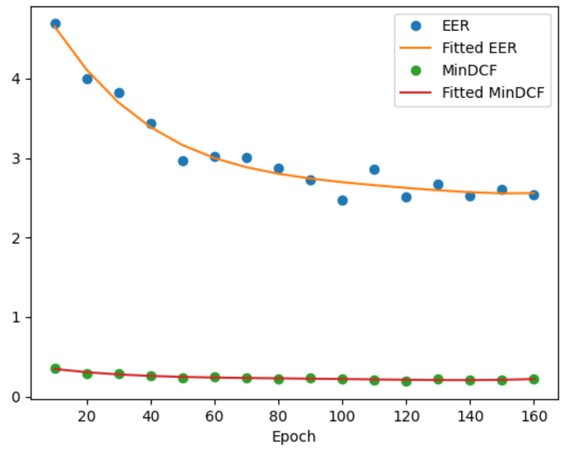


Fig. 3. Equal Error Rates (EER, %) and Minimum Decision Cost Function (minDCF) on VoxCeleb1 Test Set

10 epochs, saving time as each evaluation takes almost 10 minutes. The results show a decrease in EER from around 4.7 to 2.5, as seen by fitting a 4^{th} degree interpolated function on the evaluation points. The assumption is that additional training time would result in a further reduction in EER.

In this project work, the scores of the utilized metrics are recorded at each iteration and stored in a file for later analysis. This process of documenting the intermediate results is essential in understanding the learning process and making informed predictions for the future. It becomes especially crucial in cases of computationally intensive analyses and limited hardware resources, as it allows for a clear understanding of the trend and direction of the values.

The authors of [10] only present the final result of their experiment, which is an EER of 2.22 % on the test set. As they do not provide intermediate results, it is difficult to assess the accuracy of the reproduced training process. Hence, only the trend and progression of the learning process can be compared.

Despite this, the trend of the EER on the test set indicates that it is moving in the right direction and, given the assumptions made in this analysis, has the potential to reach the desired value of 2.22. The discrepancy between the goal value and the current result is believed to be due to the number of epochs. The authors of [10] trained for 500 epochs, while this project work only trained for 160 epochs due to the aforementioned limitations.

IV. Implementation of the Baseline System

The goal of this chapter is to determine the setup for implementing the baseline system, as described in [10]. First, the key aspects of the baseline system are analyzed and decisions are made based on performance. Then, standards for reproducibility outlined in [chapter rules] are applied to the baseline system.

1) Baseline System: The authors of [10] conduct several experiments, as described in II-B5. They keep the input representation, data, and training elements constant, but vary the models and hyperparameters. They find that the model trained with the angular prototypical function outperforms other training methods. As a result, this project work chooses to use a model with the angular prototypical function. The following list shows the key details for the implementation of the baseline system for this study:

- General Information: The system is based on a trained ResNet model which accepts two audio samples, either from the same or different speakers, compares them and produces a binary output that classifies whether the audio samples are from the same speaker.
- Input Representation: In the training phase, a fixed length 2-second segment is randomly selected from each utterance. Spectrograms are created using a 25ms hamming window and a 10ms step. The input for the Fast ResNet are 40-dimensional Mel filterbanks. Mean and variance normalization is performed through instance normalization to the network input. The VoxCeleb dataset mainly contains continuous speech, so voice activity detection is not used during training and testing.
- Model Architecture: Paper [10] presents different types of ResNet models and among them, Fast ResNet-34. Fast ResNet-34 is similar to the ResNet with 34 layers described in [10], but it uses only one quarter of the channels in each residual block to decrease computational cost. The model has 1.4 million parameters as opposed to the 22 million of the standard ResNet-34. Additionally, the input dimensions are smaller and the strides are earlier to decrease computation requirements. The performance is comparable to the ResNet model, while the computation cost is less than half of those models. Due to the hardware limitations faced in this project work, the Fast ResNet is deemed to be the most suitable option.
- Data: The dataset used is a large-scale dataset, named VoxCeleb, which comes in two versions: VoxCeleb1 and VoxCeleb2. VoxCeleb1 includes more than 150’000

utterances from 1251 celebrities, while VoxCeleb2 has over 1'000'000 utterances from 5'994 celebrities. The network is trained on the development set of VoxCeleb2 and evaluated on the test set of VoxCeleb1. It is important to note that the development set of VoxCeleb2 and the test set of VoxCeleb1 have no speakers in common. No additional data augmentation techniques are applied during training, apart from the random sampling of utterances from each identity.

- Training: The authors of [10] implement their model using the PyTorch framework. The models are trained using a NVIDIA V100 GPU with 32GB memory for 500 epochs. During training, a random selection of a maximum of 100 utterances from each of the 5'994 identities per epoch are made to address class imbalance. The Adam optimizer is used with an initial learning rate of 0.001, which decrease by 5% every 10 epochs. For metric learning objectives, the authors use the largest batch size that fit on their GPU which is 800. Due to the limited RAM capacity of 16 GB in this study, the batch size had to be decreased to 200.

2) Application of Bronze-Silver-Gold Standards: The Bronze-Silver-Gold standards are applied to the chosen baseline system.

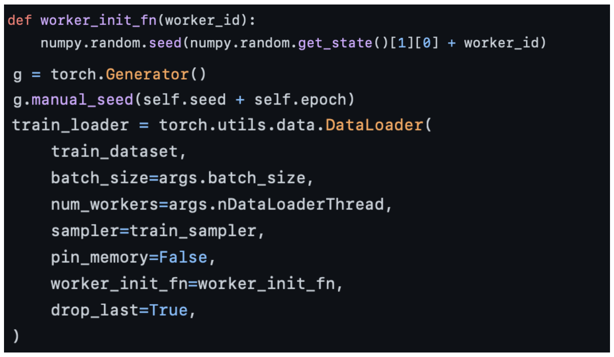
a) Bronze:

- Data: The Bronze standard requires that all datasets used in a publication be publicly accessible. [3] This includes not only the raw data, but also the information and tools needed to download and process the data. The authors of the baseline paper provide the code for downloading the VoxCeleb dataset on their GitHub repository, however the access credentials to the webpage are not included. The authors do not provide a script to preprocess the data, which has to be done during this project work. To obtain the VoxCeleb dataset, I had to contact the relevant authorities for permission to access it for educational purposes. After obtaining the necessary credentials, I was able to download the dataset. To process the dataset, I created a pipeline which involved downloading the dataset, saving it as a zip file on my local machine, verifying the download's completeness with a provided checksum, unzipping the data, and organizing it for model training and evaluation. This step is crucial as large-scale datasets may be uncertain when downloading over the internet.
- Model: A critical component of reproducibility is sharing of trained models. [3] This is because access to the model is important to determine its generalizability to other datasets, fairness in predictions, and the absence of bias or artifacts in the data. The authors of the baseline system have met this requirement by providing the implementation of their used model and the weights of the pre-trained model in a public repository. In this project, the model is saved at each iteration, also known as an epoch. This allows the training to be resumed in case of an interruption, such as a server crash. The model checkpointing strategy ensures that the process of training can continue from the last saved checkpoint, rather than starting from scratch. The saving of the checkpoint is performed at every epoch, which takes 3-4 hours with the current system and set up to complete.
- Source Code: The source code is as important as the methods section in a study. The code includes the implementation details that may not be accurately replicated from the description in the methods section. This can lead to differences in results and require a lot of effort to re-implement the entire analysis. The source code for the baseline including training, tuning, testing models and generating results has been made publicly available on GitHub. The necessary arguments for hyperparameters can be obtained from their paper. Hence, this standard has been met.

b) Silver:

- Dependencies: Reproducing a paper result using just its data, models, and code can be done by following best practices from software engineering. These include package management and managing randomness. [3]
- Package Management: A challenge encountered when trying to reproduce is the issue of incompatibility of package versions which can result in altered behavior. The difficulty lies in determining which version has been originally used in order to successfully reproduce the results. Using dependency management tools correctly can remove these challenges for both the original authors and those trying to build upon their work, as it keeps track of which versions of packages were used. [3] The authors provide a requirements.txt file to install prior the training.
  In addition to that, during this project, a Docker container has been set up. This helps managing dependencies and allows to specify the system state in which to run the code more precisely than just versions of software packages.
- Randomness: The use of randomness in machine learning includes the splitting of datasets, initialization of neural networks, and some GPU-accelerated mathematical operations in model training. As the output of models relies on these factors, it is essential to seed the pseudorandom number generators used in the analyses to maintain consistency in results. The way seeds are set varies based on the programming language, and authors must be cautious when working with deep learning libraries as they may not prioritize determinism, particularly when it comes to GPU acceleration. [3]
  The authors' system is based on Python and employs the PyTorch framework, which offers options to address the issue of non-determinism in operations. Results obtained using PyTorch will not remain consistent across different releases, commits, or platforms. Even if using identical seeds, the results between CPU and GPU executions may not be reproducible. However, it is possible to limit the

sources of non-determinism and control the sources of randomness by configuring PyTorch. This ensures that multiple calls to these operations with the same inputs will yield consistent results. [4]
The authors provide their random seed in the code base which makes it possible to reproduce the randomness. Additionally, the DataLoader is set up to reseed its workers to handle randomness in its multiprocess data loading algorithm. This is achieved by using the worker_init_fn() and generator functions as seen in figure 4.



```python
def worker_init_fn(worker_id):
    numpy.random.seed(numpy.random.get_state()[1][0] + worker_id)

g = torch.Generator()
g.manual_seed(self.seed + self.epoch)
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=args.batch_size,
    num_workers=args.nDataLoaderThread,
    sampler=train_sampler,
    pin_memory=False,
    worker_init_fn=worker_init_fn,
    drop_last=True,
)
```

Fig. 4. Data Loader with Random Seed

c) Gold: To meet the highest level of reproducibility, all steps of an analysis, including data downloading, preprocessing, model training, and output creation must be automated and reproducible with a single command. This involves tracking dependencies, making data and code available, and automating all steps. [3] The implementation of the rule has been achieved through a Python project that separates the project into different components, including dataset, training, and evaluation. The dataset component involves downloading and processing data, while the training and evaluation components train the model and evaluate it, respectively. By running the main.py file, the pipeline from data collection to the final result of the analysis can be initiated and executed without additional effort. Note: insert class diagram?

d) Compute-intensive analyses: The authors train their models using a NVIDIA V100 GPU with 32GB memory for 500 epochs. It takes them five days to train the Fast ResNet model. [10] For individuals without access to a physical GPU, a cloud server can be used instead. This project utilized the APU Cluster of the InIT, a cloud environment accessible to InIT/CAI staff.

Training the model on the server cluster takes approximately 3-4 hours per epoch, which amounts to a total of 1750 hours of computation time for the same model. This equates to roughly 10 weeks of uninterrupted training. The slower training time is due to the inefficiency of data storage and retrieval with remote hard disk storage. Since hardware limitations cannot be improved, the only option is to optimize the software. The analysis of the server cluster has revealed that the main issue lies with the data loader. This is confirmed by the observation that the GPU usage is not at full capacity when prompted through the command line.

One bottlenecks in this process is the speed of reading samples from disk. This becomes even more critical when working on the cloud, as the data has to be transferred from disk to CPU to GPU. [6] If data is loaded sequentially as shown in figure 5 the performance becomes slow.

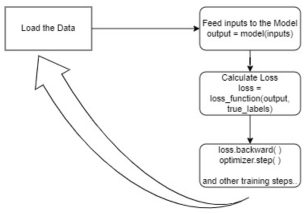

Load the Data
Feed inputs to the Model output = model(inputs)
Calculate Loss loss = loss_function(output, true_labels)
loss.backward( ) optimizer.step( )
and other training steps..

Fig. 5. Sequential Data Loading and Training

In this study's implementation, the PyTorch data loader allows for faster access to the next batch of data as soon as the current batch is processed. This is achieved by implementing a parallel data generator, as depicted in figure 6, which produces batches ready for the model to consume. The PyTorch data loader makes use of multiple subprocesses to generate data in parallel using multiple CPU cores. This way, the main process does not have to wait for data loading and can easily access the available batches of data generated by the data generator.

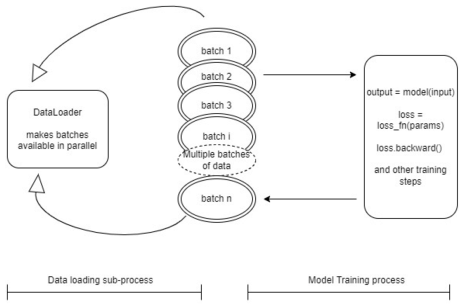


Fig. 6. Sequential Data Loading and Training

V. Conclusion

The objective of this project work is to reproduce the baseline system presented in [10]. The choice of the baseline system is made considering the availability of enough details to implement the system while taking into account the constraints of training time and computational resources.

The application of the standards to the baseline system reveals the potential gaps that can be filled with proposed methods to achieve reproducibility.

The evaluation of the implemented system demonstrates that an equal error rate (EER) of 2.5 % is obtained, which is close to the EER value of 2.22 % reported in [10]. The difference is attributed to the limited training time of 160 epochs compared to the 500 epochs used in [10].

While the reproduction of the baseline system was successful, the results highlight some limitations in the current implementation. One of the limitations is the hardware resource which results in a high training time. This limits the number of epochs to 160, compared to the 500 epochs used in [10]. This may have impacted the final EER result, as faster training time would lead to more epochs and more likely to a lower EER.

Another limitation is the lack of intermediate results provided by the authors of the paper [10], which made it difficult to determine if the metrics during learning and evaluation meet the same values. This highlights the importance of documenting intermediate results during the training process in order to have a better understanding of the learning process and make informed assumptions for future improvements.

During this study, several attempts to optimize training computation time has been done by solely adapting the software. Even though, the training time could be reduced by almost half of its time, it still falls short in comparison to the training epochs required. Mastering this task requires to know the used server cluster in detail.

Despite the limitations, this study successfully reproduces the baseline result, providing a foundation for future research and advancements in speaker verification. The proposed techniques have the potential to contribute to the development of more robust speaker verification systems in the future.

References

[1] Atlassian. Speaker recognition and verification. Available at https://wiki.aalto.fi/display/ITSP/Speaker+Recognition+and+Verification (2023/02/12).

[2] Manrai A. K. Ghassemi M. Beam, A. L. Mchallenges to the reproducibility of machine learning models in health care. Technical report, Boston, 2019.

[3] Florian Markowetz Su-In Lee Casey S. Greene Benjamin J. Heil, Michael M. Hoffman and Stephanie C. Hicks. Reproducibility standards for machine learning in the life sciences. *NaturE MEthoDs*, 18(1):1122–1144, 2021.

[4] PyTorch Contributors. Reproducibility. Available at https://pytorch.org/docs/stable/notes/randomness.html (2023/01/10).

[5] Maël Fabien. Basics of speaker verification. Available at https://maelfabien.github.io/machinelearning/basics$_speech (2023/02/12)$.

[6] Srishti Gureja. How to eliminate the data processing bottleneck with pytorch. Available at https://wandb.ai/srishti-gureja-wandb/posts/How-To-Eliminate-the-Data-Processing-Bottleneck-With-PyTorch–VmlldzoyNDMxNzM1?galleryTag=advanced (2023/01/17).

[7] Georg Heigold, Ignacio Moreno, Samy Bengio, and Noam Shazeer. End-to-end text-dependent speaker verification. *CoRR*, abs/1509.08062, 2015.

[8] Prof. Dr. H.-P. Hutter. Analysis of Sequential Data, InIT/ZHAW, November 2022.

[9] Amna Irum and Ahmad Salman. Speaker verification using deep neural networks: A review. *International Journal of Machine Learning and Computing*, 9:20–25, 02 2019.

[10] Seongkyu Mun Minjae Lee-Hee Soo Heo Soyeon Choe Chiheon Ham Sunghwan Jung Bong-Jin Lee Icksang Han Joon Son Chung, Jaesung Huh. *In defence of metric learning for speaker recognition*. PhD thesis, South Korea, 2020.

[11] Zhong Meng, Yong Zhao, Jinyu Li, and Yifan Gong. Adversarial speaker verification. *CoRR*, abs/1904.12406, 2019.

[12] Patrick Kenny Niko Brummer Pierre Ouellet Pierre Dumouchel Najim Dehak, Reda Dehak. *Support Vector Machines versus Fast Scoring in the Low-Dimensional Total Variability Space for Speaker Verification*. PhD thesis, Brighton UK, 2009.

[13] Ejiro Onose. How to solve reproducibility in ml. Available at https://neptune.ai/blog/how-to-solve-reproducibility-in-ml (2023/01/16).

[14] Pradip K. Das Pulkit Verma. *i-Vectors in Speech Processing Applications: A Survey*. PhD thesis, Guwahati, Assam 781039, India, 2021.

[15] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. X-vectors: Robust dnn embeddings for speaker recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5329–5333, 2018.