WAVEVOICE: A WAVENET BASED ARCHITECTURE FOR SPEAKER VERIFICATION

Daniel Neururer

ZHAW Datalab
Zurich University of Applied Sciences
Winterthur, Switzerland

`neurudan@students.zhaw.ch`

ABSTRACT

In this paper we discuss a modified architecture of the WaveNet applied to speaker verification, which is called WaveVoice. The models were trained on VoxCeleb2, where raw audio waveform as well as mel frequency spectrograms extracted from the audio have been used. All models have been evaluated on VoxCeleb1 with the EER metric. Moreover, a novel method to train a feature extraction model has been analyzed. This method has been named the hyperepoch setting. Furthermore we want to show the achieved improvements regarding computational efficiency, where we were able to reduce the training time of our chosen baseline from ∼3 years to ∼8 days on our available infrastructure.

1 Introduction

Speaker verification [1] is a sub-area of speaker recognition, where two utterances are being compared and the goal is to predict the likelihood of them being from the same speaker. Deep Neural Networks that have been trained first on supervised speaker recognition [2][3][4][5], have proven to perform well on speaker verification, as those models can be used to extract features that should represent the voice of a speaker.

In earlier work [3] we came to the conclusion that the raw audio waveform may contain some information that get lost when they are being transformed into a spectrogram. This information may not seem important to a human, but could still be of value for a neural network.

The WaveNet architecture as introduced by Oord et al. [6] is a fully convolutional deep neural network for the generation of speech in the form of raw audio. The training works by feeding the model a segment of speech and letting it predict the next following timestep of that segment. When being conditioned on text, it performs the task of text to speech generation with great naturality. Also, when conditioned on speakers, it is able to model speech that sounds incredibly close to the conditioned speaker. This led us to the idea of introducing it to speaker verification as well. As the WaveNet is not bounded to raw audio, we also discuss and compare the performance of the architecture using mel frequency spectrograms.

2 Related Work

With the release of the two large-scale datasets VoxCeleb1 [7] and VoxCeleb2 [8], speaker verification has gained in popularity. Xie et al. [2] have shown that with a thin-ResNet as a frontend architecture, an EER of 3.22 % can be achieved on VoxCeleb1. Their approach has been trained on spectrograms and have set the new state of the art.

Closer to our architecture, the RawNet [5] as well as the SincNet [9] have both been designed to perform the task on raw audio. The SincNet applies frequency band pass filters in the waveform domain using the initial convolutional layer and yielded promising results. However, it has only been trained on the TIMIT [10] and LibriSpeech corpus [11] and thus is not well compareable to our setting. The RawNet on the other hand comprises residual blocks, a gated recurrent unit layer and fully connected layers for the embeddings and output. This model has set the state of the art with an EER of 4.0 % in speaker verification when training on raw audio.

3 Experimental Setup

3.1 WaveVoice

We propose a modified version of the WaveNet architecture [6], namely WaveVoice. Experiments using raw audio data as well as mel frequency spectrograms have been executed to test the performance of the model. In the following sections we will discuss our proposed architecture, evaluation metrics and show the achieved results compared to our baseline.

3.1.1 Model Architecture

The WaveVoice architecture differs from the WaveNet in terms of causality, as future timesteps are not to be depended

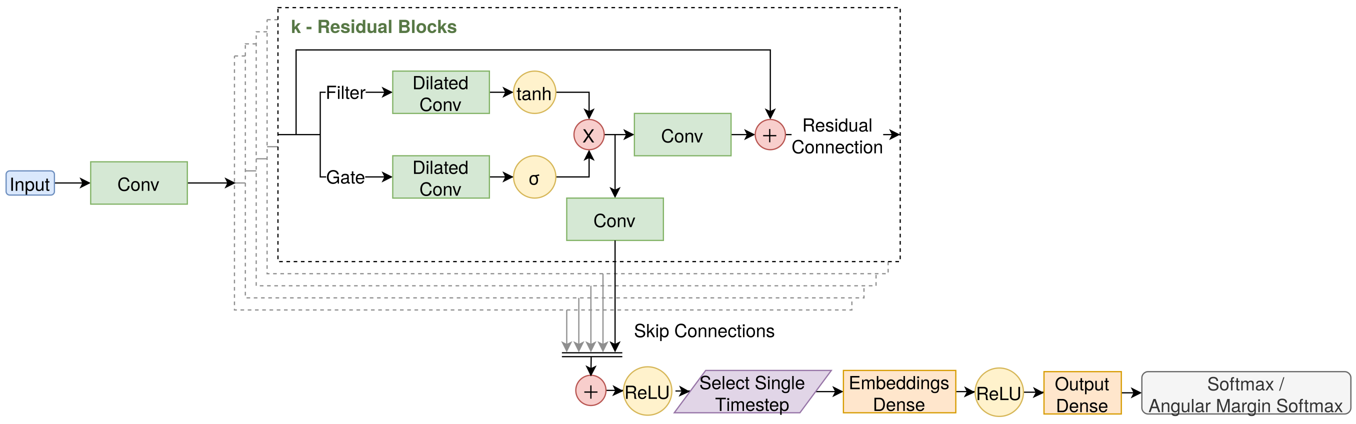

Fig. 1. WaveVoice Architecture - In our setting, only the acausal model which selects the middle timestep has been tested. Here, ”Dense” blocks represent fully connected layers and ”Conv” blocks 1D convolutional layers.

on anymore. Furthermore, the output layers of the network have been changed. The whole architecture can be seen in detail in Figure 1.

3.1.1.1 Dilated Convolutional Layer

The basic foundation of the WaveNet [6] and WaveVoice architecture are the dilated causal convolutions. The dilation of a convolution means, that it skips a certain amount of timesteps defined by the dilation rate. E.g., if we set the dilation rate of 2, only every second kernel weight is not set to zero. With a dilation rate of 4, only every fourth kernel weight is not set to zero and so on [12]. If multiple layers are stacked together, while exponentially increasing the dilation rate for each layer, the receptive field is increasing exponentially as well.

3.1.1.2 Gated Activation Units

Instead of using a simple ReLU activation function, the WaveNet takes advantage of gated activation units, as it has been proven to be beneficial for the performance in earlier work [13].

$$\mathbf{z} = \tanh(W_{f,k} * \mathbf{x}) \odot \sigma(W_{g,k} * \mathbf{x}) \tag{1}$$

In Equation 1, $W_{f,k}$ and $W_{g,k}$ are the weights of the filter and the gate convolution respectively. k stands for the layer index, which determines the dilation rate b^k of the convolution layers, with b being the dilation base. $*$ stands for the convolution operator and $\odot$ for the elementwise multiplication operator.

Each unit consists of a filter and a gate, where the filter is intended to modify the input and the gate determines how much of the modified input should actually go through the unit. Both filter and gate are made out of a dilated causal convolution, which then get activated by a hyperbolic tangent and a sigmoid activation function respectively. Finally, the output of filter and gate are being elementwisely multiplied together, forming the output of the unit.

3.1.1.3 Residual Blocks

So called residual blocks are used in the WaveNet and WaveVoice, similar to the ResNet architecture first introduced by He et al. [14]. Each block consists of a gated activation unit followed by two separate convolutional layers. One of them will afterwards be added to the input of the block and thus forms the residual connection and will be fed into the next block. The other one results in a skip connection, which is being stored temporarily. The final output of the blocks will be the sum of all skip connections.

3.1.1.4 Receptive Field

The WaveNet [6] architecture as mentioned in the paper is designed to predict timesteps in the future. Therefore the dilated convolutional layers have to be causal. However, we do not try to predict future timesteps in our setting anymore. This gives us the possibility to use acausal dilated convolutional in our architecture, similar to the encoder part of the ByteNet [15] architecture. By using a filter size of 3, we are able to encode the information of a segment into the timestep in the middle instead of the last one.

As we have a sampling frequency of 16kHz for the raw audio waveforms, we want to increase our receptive field as much as possible. So instead of using a dilation base of 2 like in the WaveNet [6] and ByteNet [15], we set it to 3.

The receptive field can be calculated by using the following equation:

$$\mathrm{RF} = b^d + \sum_{i=0}^{d-1} b^i * (f - b) \tag{2}$$

where b and d stand for the dilation base and number of residual blocks respectively, and f denotes the filter size.
Five residual blocks have been used for our models trained on mel frequency spectrograms, leading to a receptive field of 243 (2.43 seconds). When switching to raw audio waveforms, a maximum of eight residual blocks resulting in a receptive field of 6’561 ($\sim$0.410 seconds) were applied. Due to a lack

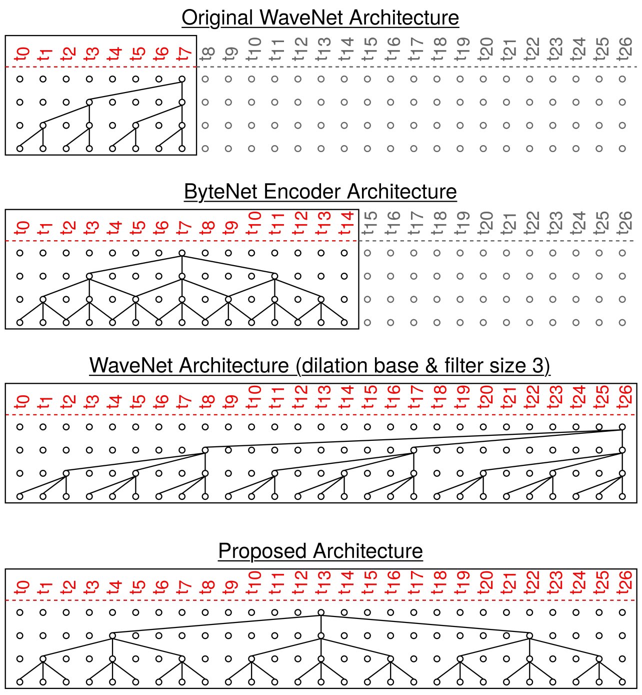
Original WaveNet Architecture
ByteNet Encoder Architecture
WaveNet Architecture (dilation base & filter size 3)
Proposed Architecture
t0 … t26

Fig. 2. Comparison of the receptive field (RF) for the original WaveNet architecture (RF $=$ 8), the ByteNet encoder (RF $=$ 15), the WaveNet architecture with a dilation base of three (RF $=$ 27) and our proposed approach (RF $=$ 27) for three residual blocks each.

of computational resources and time, we were not able to perform additional experiments using more residual blocks and different sizes.

3.1.1.5 Training Setting As the primary research question was to analyze if the WaveNet [6] architecture was appropriate for speaker verification tasks, WaveVoice has been designed to resemble the original structure as much as possible. Since our trained model eventually needs to extract features compressed along the time axis, we select the timestep where the temporal information is encoded in. Subsequently, we feed them into a fully connected layer, called the embedding layer.
The model is fist trained to predict the matching speaker of an utterance. The output of the embedding layer is then believed to be able to represent a voice as expressive as possible. Initially, the use angular margin softmax losses [16][17][18] was intended, as earlier work [3][4][2] proved them to be useful for speaker verification as well. However, due to the fact that it needs a lot more hyperparameter tuning, and training a model typically takes several days, we have decided to train our models using the standard softmax loss.

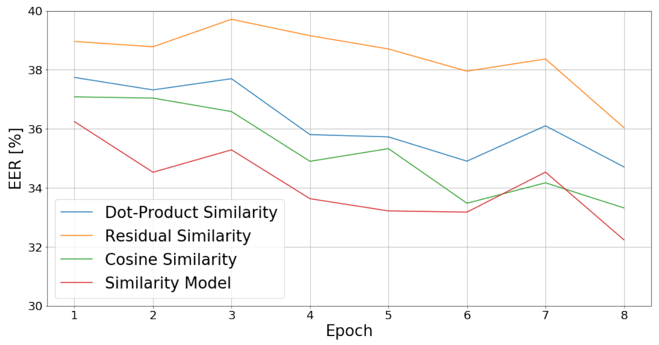

Fig. 3. EER values of the tested similarity methods. We have used the proposed WaveVoice architecture with a filter size of 64, embedding filter size of 128 and 8 residual blocks and trained it on the raw audio waveforms using the hyperepoch setting.

3.1.2 Evaluation

To evaluate the performance of our architecture, the model is cut after the embedding layer and the extracted features are used to compare two utterances with each other using a similarity measure. With the resulting scores, the equal error rate (EER), which is the commonly used evaluation metric in speaker verification, is then calculated. As our proposed architecture is only able to extract the features of short segments with a fixed length, an utterance is splitted into parts of that length and fed to the model. Since this results in multiple features, we calculate the mean feature values of all parts.
We have compared the effect on the performance of the following four similarity measures, where $\vec{A}$ and $\vec{B}$ denote the extracted feature vectors of two utterances, N the length of the feature vectors and s the resulting similarity score.

3.1.2.1 Dot-Product Similarity

$$s(\vec{A},\vec{B}) = \vec{A}\cdot\vec{B} \tag{3}$$

As used by Xie et al. [2], one can simply calculate the dot product of $\vec{A}$ and $\vec{B}$ and use it as a score. The lower boundary of the score is 0, as a ReLU activation in the embedding layer is used. However, it has no upper boundary which means that decision thresholds have to be computed for the scores when the EER is calculated. Therefore, during inference it is impossible to decide if $\vec{A}$ and $\vec{B}$ are from the same speaker, if the thresholds have not been stored at evaluation time and either only two utterances, strictly utterances from the same speaker or strictly from different speakers are being compared.

3.1.2.2 Residual Similarity

$$s(\vec{A},\vec{B}) = -\sum_{i}^{N}\left|\vec{A}_i - \vec{B}_i\right| \tag{4}$$

Another idea was to take the negative of the sum of the absolute difference from $\vec{A}$ and $\vec{B}$. With this measure, the sum of all absolute distances between each feature can be calculated. It is important that the result is then negated, as a small distance between the features means that the values are more similar to each other than when they have a large distance. Still, this method has the same drawback as the dot product similarity, with the difference that the upper boundary is 0 and the lower boundary $-\infty$.

3.1.2.3 Cosine Similarity

$$s(\vec{A},\vec{B}) = \frac{\vec{A}\cdot\vec{B}}{\left\|\vec{A}\right\|\left\|\vec{B}\right\|} \qquad (5)$$

Cosine similarity measures the cosine of the angle between two vectors in an inner product space. This metric has proved to beat the two previously mentioned methods and additionally yields scores between -1 and 1. In our case as we have strictly positive results, the scores will range from 0 to 1.

3.1.2.4 Similarity Model We assume, that depending on the mood or health situation of a speaker in an utterance, the resulting feature vectors may look completely different. E.g., let us consider two utterances of one speaker, where he has a sore throat and is in a delirious mood in one of them and healthy and in a euphoric state in the other. We expect dissimilarities of some expressive features. Those changes do not necessarily have an effect on the training, as the model can learn the pattern of those states. Therefore, it will still be able to predict the correct speaker. However, using simple mathematical similarity measure, one may not be able to associate the features to the same speaker anymore.

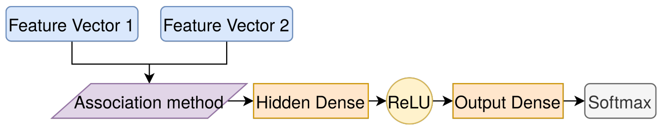

Fig. 4. Similarity Model architecture. The ”Association method” block can be selected as pleased (concatenate, sum, multiply, etc.). In our approach, we have only tested the sum of the two feature Vectors.

Therefore, we have trained a simple MLP that learns if two feature vectors originate from the same speaker, and used the prediction scores as our similarity score. The similarity model needs to be trained from scratch for 100 epochs on the whole VoxCeleb2 [8] dataset every time the front-end architecture is evaluated. Thus, we have only been able to apply the model on eight evaluation steps, due to time constraints. The resulting EERs are shown in detail in Figure 3.

Albeit having only eight values, it is clearly visible that the EER can be typically reduced when switching to the similarity model. Compared to the cosine similarity, we have a mean decrease of 1.13%, which leads to the assumption that there could be some potential in the measure.

3.1.3 Results

We have chosen the results of Jung et al. [5][19] and the results of Xie et al. [2] as a baseline to compare our models trained on raw audio and on mel frequency spectrograms with, respectively. (results are shown in Table 1)

3.2 Hyperepoch Setting

A big problem when training a DNN to extract features, is that it cannot actively be controlled what the features should represent. E.g., when looking at speaker verification, we first train our model on utterances from a large set of speakers to classify the corresponding speaker. This will lead the model to learn how to represent the speakers in the dataset.

```
==================================================
    ------------------
    # initial training
    ------------------
1.  spkrs = sample N random speakers
2.  train model on spkrs for #train_epochs

    ---------------------
    # Hyperepoch training
    ---------------------
    for hyperepoch in range(#hyper_epochs):

       -------------------
       # pretraining phase
       -------------------
3.     spkrs = sample N random speakers
4.     reset weights of output dense layer
5.     make only output dense layer trainable
6.     train model on spkrs for #pretrain_epochs

       ------------------------
       # feature training phase
       ------------------------
7.1    make whole model trainable
7.2    make whole model trainable, except last
       layer

8.     train model on spkrs for #feat_train_epochs
==================================================
```

Listing 1. Hyperepoch Setting Pseudocode. Step 7.1 is used in the first variant of the setting, where we train the whole network including the output layer in the feature training phase, and use step 7.2 for the second variant, where we do not train the output layer anymore.

However, since our task requires us to learn how to represent any voice, rather than learning how to represent a speaker in the dataset, we have came up with an original way to train

	Input Features	Frontend	Backend	Loss	EER (%)
Jung et al. [19]	Raw waveform	CNN-LSTM	Cosine Sim.	Softmax	8.7
Jung et al. [19]	Raw waveform	CNN-LSTM	b-vector	Softmax	7.7
Jung et al. [5]	Raw waveform	RawNet	Cosine Sim.	Softmax+Center+BS	4.8
Jung et al. [5] (baseline)	Raw waveform	RawNet	Concat+Mul	Softmax+Center+BS	**4.0**
Ours	Raw waveform	WaveVoice	Cosine Sim.	Softmax	24.4
Ours	Raw waveform	WaveVoice	Dot-Prod Sim.	Softmax	30.9
Ours	Raw waveform	WaveVoice	Residual Sim.	Softmax	28.9
Xie et al. [2] (baseline)	Spectrogram	Thin ResNet-34	Dot-Prod Sim.	Softmax	**3.22**
Xie et al. [2]	Spectrogram	Thin ResNet-34	Dot-Prod Sim.	AM-Softmax	3.23
Jung et al. [5]	Mel-Spectrograms	i-vector	PLDA	-	5.1
Ours	Mel-Spectrograms	WaveVoice	Cosine Sim.	Softmax	30.9
Ours	Mel-Spectrograms	WaveVoice	Dot-Prod Sim.	Softmax	34.6
Ours	Mel-Spectrograms	WaveVoice	Residual Sim.	Softmax	32.3

Table 1. Comparison of the results for speaker verification on the original VoxCeleb1 test set. We did not see it as necessary including the latest state of the art results from VoxSRC [20], as our results did by no means match up with the chosen baselines.

the model, which we call the hyperepoch setting.
Usually when a model is retrained on new labels, it will forget some information, that it has learned before. Still, the learned information should be kept and the association to a specific label forgotten. This is why the output layer is trained first when starting a hyperepoch. By doing so, it should learn how to predict the speaker based on the features the model has learned so far. When starting the feature training, the output layer already knows how to associate the features with the speakers.
We tried two variants of training the feature extractor. The first one will train each layer, whereas the second one will only train the layers up to the feature extraction fully connected layer. By applying this setting, we try to force the model to separate the information it knows about the speakers to only persist in the output fully connected layer and what it knows about how a voice can be represented in the feature extractor part of the desired architecture. To check how much a model profits from the setting, the pretrain accuracy of the model after each hyperepoch can be analyzed, as well as by simply comparing its EER to the achieved EER of a saturated model that has been trained on all speakers without the hyperepoch setting.

Results showed that the model for both variants will in fact improve in terms of accuracy of the pretraining phase after each hyperepoch (see Figure 6). However, compared to the baseline we notice a typically larger fluctuation of the EER, which furthermore is also larger. This leads to the conclusion that the proposed new setting does not yield the originally intended effect on the model. A possible reason for the negative effect of the hyperepoch setting could be, that we cannot make use of the full potential of the used Adam optimizer anymore, since the stored moving averages of the gradients have to be reset at the beginning of every hyperepoch. That being said,

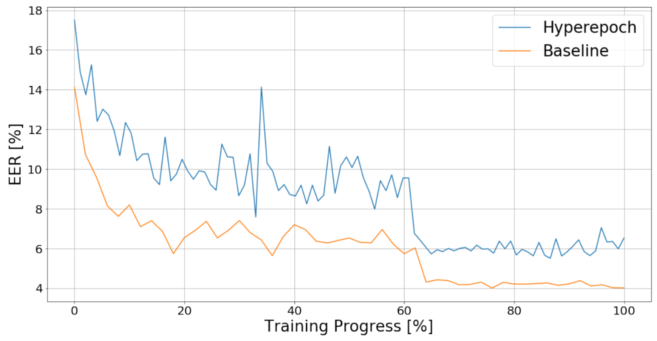

Fig. 5. EER of the hyperepoch setting applied to the baseline model by Xie et al. [2], compared to the original baseline. As the hyperepoch setting typically needs more training epochs and not being finished, we have decided to show the training progress in percentage of both models, with 100% being the last state of the hyperepoch model.

the more obvious reason seems to be that the model keeps unlearning most of the information that it has already learned.
The setting has been tested on the architecture proposed by Xie et al. [2]. However, as training uses 15 days upwards, we did not finish the final calculation of the model in time.

4 Speed Improvement

The WaveVoice architecture by itself is not all too complex if we compare it to our baseline by Xie et al. [2] who have ∼11 million trainable parameters, our largest architecture using 128 filters for the embedding layer and all convolutional layers only takes ∼1.8 million trainable parameters. However, as we use four convolutional layers per residual block, with the input of each always having the same length of 6’561 or 243, we have a massive amount of gradients for a single

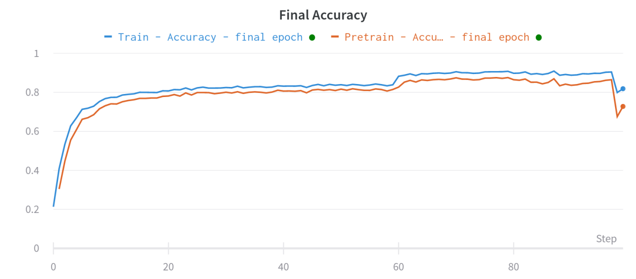

Fig. 6. Pretraining (orange) and feature training (blue) accuracy curve of the hyperepoch setting applied to the baseline model.

sample during training. This has a drastic influence on the batchsize we are able to use, as the memory of our available GPU's is limited to ∼11.4 GB. Thus, our model was rather quick in terms of calculation and therefore was limited to the speed we were able to load the data.

Using the framework [21] of the work by Xie et al. [2], the audio files are being read and preprocessed in 10 separate threads. As the storage system of our GPU-cluster is not located on the same machine as the GPU itself, we have to transfer the data over ethernet, slowing us down already considerably. Moreover, the storage devices are not designed to repeatedly read a huge amount of files efficiently. Thus, for a single epoch of training the model using this framework, we would have needed ∼8 days to train a single epoch. As the standard setting in the framework were 128 epochs, the training would have taken ∼3 years.

In earlier work we have created datasets using the pickle module of python which is fast as long as our whole dataset fits in the available RAM of 64 GB. However, the extracted VoxCeleb2 [8] dataset is ∼76 GB large for the raw audio waveforms and even larger when loaded into a numpy array. This is why we would need to split it into subfiles and continuously read parts of the dataset. Since pickle uses object serialization, the disk space required for the data would increases usually by a factor of two.

Those problems led us to the need of a more efficient way to store and read our data. We came to the conclusion that the HDF5 format solves most of our problems the best. The format uses a binary data pipe, enabling us to decrease the disk space used compared to pickle. We can look at the format as a form of a file system, with a group being the equivalent to a folder and a dataset being a data storage. To simplify readability and handling for later usage, we have decided to create a group for the actual data, one for storing the names of the audio files separately as well as one that contains statistical information about the audiofiles. For all of those three groups, we create a dataset for each speaker. So instead of searching through the whole dataset for a specific sample, we first access the speaker dataset and search through a smaller space, which proved to increase the access speed to a specific sample furher. Another major benefit of HDF5 is the ability of slicing. To read a defined part of specific sample, we do not have to load a whole dataset into our RAM, but can exactly access that part of the sample.

The dataset with the statistical data contains up to now only the length of an audiofile, as well as the total length of all audiofiles from a speaker. We intend to add more statistical information in future work. Because of all the additional information and the data being not compressed in a specific format anymore, the whole processed dataset takes ∼560 GB of disk space and takes up to 8 days to generate.

During training, we have noticed that we still were not able to use the full potential of our GPU performance as we have had an average utilization percentage of ∼10 %. To improve our reading speed, we decided to load the lengths of each audiofile in advance and calculate the start and end of each utterance of the train and validation part and store that information together with the speaker id and location of the utterance in the dataset in a list which we call the location list. We can now calculate a random slicing range for the samples without having to load any further information from our dataset and thus can draw samples much faster. Doing this, the GPU utilization has been increased to a mean of ∼18 %.

Eventually, we have implemented a multi-threaded data generator, consisting of an indexer and several enqueuers. The indexer shuffles the location list at the beginning of an epoch and splits the location list into batches. It then feeds all those batches into an index queue. Each enqueuer initially opens the dataset file and then repeatedly reads out a batches from the index queue and then draws the data and feeds it to a sample queue. It is important that the file handler is being left open, since this further decreases the reading times. We then only have to read out batches from the sample queue and train our model on it.

Using all above mentioned modifications with up to 100 enqueuer threads, we were able to decrease the reading time for a batch of 80 samples from ∼55 seconds to ∼50 milliseconds. In the end, to train our models we always had a mean GPU utilization percentage of ∼90 %, which seems to be the limitation of them. Thus, the training time decreased from the initial ∼3 years to ∼8 days.

5 Conclusions

As the correct implementation of the speed improvement part took the most of the available time, we were not able to perform many experiments since most of them still take several days to complete. The models trained on the mel frequency spectrograms usually showed a the phenomenom of learning to forget, as visible in Figure 8. However, we are not certain if the WaveVoice architecture could have performed better on

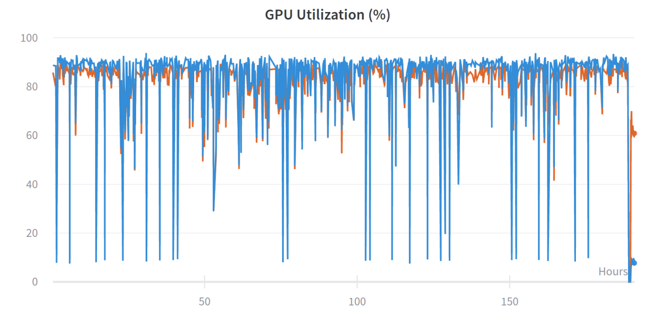

Fig. 7. GPU utilization of two separate GPU's while training the baseline model by Xie et al. [2]. Before applying the speed improving modifications, both of the curves were on the same level as the x axis.

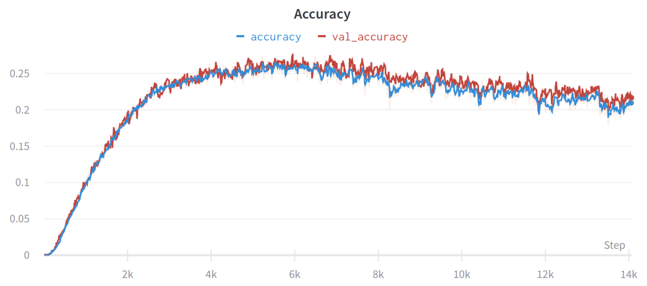

Fig. 8. Accuracy curve of the WaveVoice model being trained on mel frequency spectrograms.

the raw audio waveform data, as there seemed to be room for improvement in some runs. With some further experiments, the architecture may lead to a better performing model.

One of the possible reasons for the bad performance is that we did not properly preprocess our input data. In hindsight, we should have normalized the data appropriately. Also, adding some sort of data augmentation could have improved performance. We see the augmentation steps introduced by Povey et al. [22] as promising, which have recently been reused by Karafiát et al. [23] in automatic speech recognition. Furthermore, we did not apply voice activity detection (VAD) to our data and may have a lot of silent or noise parts in it. This should not have that big of an influence on the models trained using mel frequency spectrograms, as their input is 2.43 seconds long. The models trained on the raw waveforms on the other hand easily could encounter a sample containing only silence or noise, leading to possible confusion.

As we take a look at the hyperepoch setting, we can savely say that we did not achieve the intended effect. Some of the runs even encountered a time of catastrophic forgetting.

6 Future Work

With the preparation and proper handling of a dataset always being relatively time consuming, we see the idea of implementing a framework to simplify those tasks as a great help for future projects.

The WaveVoice architecture is currently not in a state where we can say much about how much we could take profit of. We see room for improvement in the structure of the architecture, especially when applied on raw audio waveforms. Additional experiments using angular margin softmax losses [16][17][18] could lead to an increase in performance and further understanding about structures of generated features.

We see the idea of adding SincNet filters [9] on the initial convolutional layer as promising. This would let us have only certain filter bands in the waveform domain, and proved to be beneficial for speaker recognition and verification tasks on raw waveforms.

Furthermore, we would like to perform a detailed statistical analysis of the extracted features during training, as it could lead to new insights of how we can best compare them. Methods to transform them into a lower dimensional space like PCA [24] or t-SNE [25] could help us gaining more knowledge about how certain features influence the EER.

As the similarity model has shown a clear beneficial effect on the EER, performing additional experiments with it seems interesting as well. Finally, we see the implementation of the kaldi augmentation [22][23] as promising.

7 References

[1] Homayoon Beigi, "Speaker recognition," in *Fundamentals of Speaker Recognition*, pp. 543–559. Springer, 2011.

[2] Weidi Xie, Arsha Nagrani, Joon Son Chung, and Andrew Zisserman, "Utterance-level aggregation for speaker recognition in the wild," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 5791–5795.

[3] Daniel Neururer, Claude Lehmann, Patrick Walter, Jan Sonderegger, and Thilo Stadelmann, "Anglevoice: Leveraging angular margin losses for real world speaker recognition, clustering and diarization," 2019.

[4] Claude Lehmann and Thilo Stadelmann, "Real-world speaker recognition on voxceleb2 using angular margin losses," 2020.

[5] Jee-weon Jung, Hee-Soo Heo, Ju-ho Kim, Hye-jin Shim, and Ha-Jin Yu, "Rawnet: Advanced end-to-end deep neural network using raw waveforms for

text-independent speaker verification," *arXiv preprint arXiv:1904.08104*, 2019.

[6] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[7] A. Nagrani, J. S. Chung, and A. Zisserman, "Voxceleb: a large-scale speaker identification dataset," in *INTERSPEECH*, 2017.

[8] Joon Son Chung, Arsha Nagrani, and Andrew Zisserman, "Voxceleb2: Deep speaker recognition," *arXiv preprint arXiv:1806.05622*, 2018.

[9] Mirco Ravanelli and Yoshua Bengio, "Speaker recognition from raw waveform with sincnet," in *2018 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2018, pp. 1021–1028.

[10] John S Garofolo, Lori F Lamel, William M Fisher, Jonathan G Fiscus, and David S Pallett, "Darpa timit acoustic-phonetic continous speech corpus cd-rom. nist speech disc 1-1.1," *NASA STI/Recon technical report n*, vol. 93, 1993.

[11] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur, "Librispeech: an asr corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 5206–5210.

[12] Sik-Ho Tsang, "Review: Deeplabv3 — atrous convolution (semantic segmentation)," *https://towardsdatascience.com/review-deeplabv3-atrous-convolution-semantic-segmentation-6d818bfd1d74*, Jan 2019, accessed 2020-02-12.

[13] Aaron Van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al., "Conditional image generation with pixelcnn decoders," in *Advances in neural information processing systems*, 2016, pp. 4790–4798.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu, "Neural machine translation in linear time," *arXiv preprint arXiv:1610.10099*, 2016.

[16] Jiankang Deng, Jia Guo, Niannan Xue, and Stefanos Zafeiriou, "Arcface: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4690–4699.

[17] Hao Wang, Yitong Wang, Zheng Zhou, Xing Ji, Dihong Gong, Jingchao Zhou, Zhifeng Li, and Wei Liu, "Cosface: Large margin cosine loss for deep face recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5265–5274.

[18] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song, "Sphereface: Deep hypersphere embedding for face recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 212–220.

[19] Jee-Weon Jung, Hee-Soo Heo, IL-Ho Yang, Hye-Jin Shim, and Ha-Jin Yu, "Avoiding speaker overfitting in end-to-end dnns using raw waveform for text-independent speaker verification," *extraction*, vol. 8, no. 12, pp. 23–24, 2018.

[20] Joon Son Chung, Arsha Nagrani, Ernesto Coto, Weidi Xie, Mitchell McLaren, Douglas A Reynolds, and Andrew Zisserman, "Voxsrc 2019: The first voxceleb speaker recognition challenge," *arXiv preprint arXiv:1912.02522*, 2019.

[21] Weidi Xie, "VGG-Speaker-Recognition, Keras implementation," *https://github.com/WeidiXie/VGG-Speaker-Recognition*, 2019, accessed 2020-02-11.

[22] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al., "The kaldi speech recognition toolkit," in *IEEE 2011 workshop on automatic speech recognition and understanding*. IEEE Signal Processing Society, 2011, number CONF.

[23] Martin Karafiát, Murali Karthick Baskar, Igor Szöke, Hari Krishna Vydana, Karel Veselỳ, Jan Černockỳ, et al., "But opensat 2019 speech recognition system," *arXiv preprint arXiv:2001.11360*, 2020.

[24] Svante Wold, Kim Esbensen, and Paul Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[25] Laurens van der Maaten and Geoffrey Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.