PROCEEDINGS of the
23rd International Congress on Acoustics
9 to 13 September 2019 in Aachen, Germany

Real-time audio processing on a Raspberry Pi using deep neural networks

Fotios Drakopoulos[(1)], Deepak Baby[(2)], Sarah Verhulst[(3)]

[(1)]Ghent University, Belgium, fotios.drakopoulos@ugent.be

[(2)]Ghent University, Belgium, deepak.baby@ugent.be

[(3)]Ghent University, Belgium, s.verhulst@ugent.be

ABSTRACT

Over the past years, deep neural networks (DNNs) have quickly grown into the state-of-the-art technology for various machine learning tasks such as image and speech recognition or natural language processing. However, as DNN-based applications typically require significant amounts of computation, running DNNs on resource-constrained devices still constitutes a challenge, especially for real-time applications such as low-latency audio processing. In this paper, we aimed to perform real-time noise suppression on a low-cost embedded platform with limited resources, using a pre-trained DNN-based speech enhancement model. A portable setup was employed, consisting of a Raspberry Pi 3 Model B+ fitted with a soundcard and headphones. A (basic) low-latency Python framework was developed to accommodate an audio processing algorithm operating in a real-time environment. Various layouts and trainable parameters of the DNN-based model as well as different processing time intervals (from 64 up to 8 ms) were tested and compared using objective metrics (e.g. PESQ, segSNR) to achieve the best possible trade-off between noise suppression performance and audio latency. We show that 10-layer DNNs with up to 350,000 trainable parameters can successfully be implemented on the Raspberry Pi 3 Model B+ and yield latencies below 16-ms for real-time audio applications.

Keywords: Real-time, Audio processing, Low-latency, Deep Neural Networks, Raspberry Pi

1 INTRODUCTION

Speech enhancement aims to improve the quality and intelligibility of speech by suppressing adverse components such as background noise or room reverberation. This enhancement has numerous applications, including the improvement of mobile communications, speech recognition and speaker identification systems [1, 2]. At the same time, speech enhancement techniques have been adopted by hearing aids and cochlear implants and are used together with gain prescription strategies to reduce discomfort and increase intelligibility [3].

Numerous noise suppression approaches have been proposed over the past years [4] such as spectral subtraction [5, 6], Wiener filtering [7] or even psychoacoustically-based methods [8]. However, deep neural network (DNN) based approaches have quickly grown into the state-of-the-art technology due to their ability to learn complex functions from large example sets. These novel systems have been shown to outperform most of the conventional approaches [9, 10], but many DNN approaches are still based on time-frequency (T-F) masking for speech enhancement. A T-F mask is typically estimated, using short-time Fourier analysis/synthesis framework, which provides the amount of noise present in each short time-frequency point. This approach only modifies the magnitude spectrogram of the signal and ignores the phase mismatch between the noisy and clean speech signals.

However, preserving the phase of the speech signal has been shown to be important for speech quality [11]. For this reason, this paper focuses on a speech enhancement method which both preserves the signal phase and is DNN-based. To this end, we adopt the use of DNN-based auto-encoder speech enhancement systems which can directly map the raw noisy speech waveform to the underlying clean speech waveform. In contrast to most of the current techniques, the auto-encoder systems operate at the waveform level of the signal, training the model end-to-end. Recently, these architectures have been widely adopted in DNN-based

models and have shown remarkable performance in speech enhancement applications [12, 13], while being simple enough in terms of computation. This mak es them perfectly suitable for implementations in devices with limited resources.

This paper presents a portable low-cost platform for real-time audio processing, which provides a low-latency audio environment suitable for real-time signal processing algorithms. Different DNN layouts are implemented and evaluated, to study to what extend real-time speech enhancement can be performed on a device with limited resources. The ultimate goal is to design a platform where DNN-based systems can be tested in realistic scenarios, while yielding latencies below 10 ms for real-time audio applications such as hearing aids.

2 REAL-TIME SPEECH ENHANCEMENT USING DEEP NEURAL NETWORKS

2.1 Auto-encoder convolutional neural networks

The enhancement problem comprises the processing of a noisy signal y to yield an estimation of the original clean speech signal x. We propose the use of auto-encoder convolutional neural networks (AECNN) to perform this speech enhancement. These models utilize an encoder-decoder architecture (Figure 1). The input is a frame of the noisy speech signal y and the output is the enhanced signal $\hat{x} = G(y)$.

In the encoding stage G_{enc}, the input signal is projected and compressed, to yield a convolution result out of every N steps of the filter. Decimation is applied until a condensed representation $y_c = G_{enc}(y)$ is obtained. The G network also features skip connections, connecting each encoding layer to its respective decoding layer, to avoid the loss of many low level details through the encoder compression. Skip connections directly pass the fine-grained information of the waveform to each decoding stage (e.g. phase, alignment) to reconstruct the speech waveform properly. G is composed of one-dimensional strided convolutional layers of a fixed filter width w and strides of $N = 2$. The amount of filters per layer increases so that the depth gets larger as the width (signal samples L_i) becomes narrower. The resulting dimensions per layer (samples $\times$ feature maps) are computed for each kernel dimension by decreasing the signal width of the previous layer L_{i-1} to half and applying a set of k_i filters on top. The decoder stage of G is a mirroring of the encoder with the same filter widths number of filters per layer. However, skip connections make the number of feature maps in every layer to be doubled.

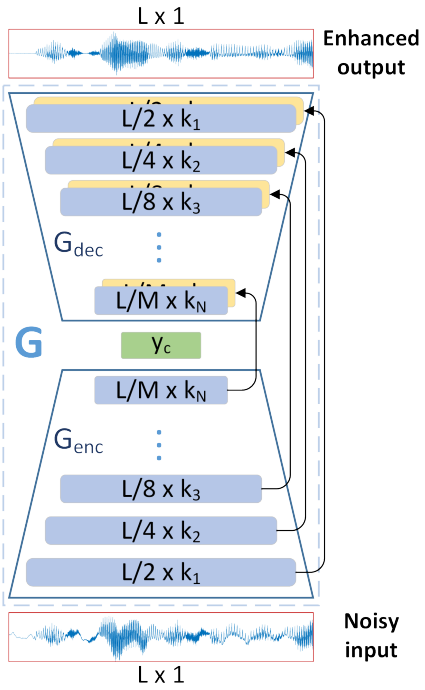

Figure 1. Encoder-decoder architecture for speech-enhancement DNN-based models. The arrows between encoder and decoder blocks represent skip connections, which pass the necessary information to the respective latent layers.

All the layers of the encoder and decoder networks are trained end-to-end, so that all processing steps are performed at the waveform level of the signal. The G network is designed to be simple enough, using only fully convolutional layers. This enforces the network to train focusing on temporally-close correlations in the input signal. Furthermore, it reduces the number of trainable parameters and hence the execution time.

2.2 Real-time framework and low-resource platform

For the development of the real-time audio processing platform a portable setup was employed, consisting of a Raspberry Pi 3 Model B+. The Raspberry Pi 3 Model B+ is equipped with a Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit SoC @ 1.4GHz, a Broadcom Videocore-IV GPU and 1GB LPDDR2 SDRAM. A HAT low-latency soundcard and earphones with a microphone were connected to the Raspberry Pi.

A low-latency framework was designed to accommodate audio processing algorithms operating in a real-time environment. Python was used as the programming environment, since it offers a straightforward interface

for hosting DNN-based models. JACK Audio Connection Kit (JACK)[1] was chosen as the audio server, due to its capability to provide real-time audio with up to 2 ms latency. The JACK-Client module[2] was used to properly bind the JACK library to the Python environment. The developed Python framework is available on GitHub[3].

The framework was designed to support overlapping windows for audio input processing and can use frame buffering to provide the necessary temporal context (resolution) while keeping the latency short. The benefit of this design is that, given a DNN-based model with a fixed input/output frame size, smaller frames can be used for the audio server (for instance $\frac{1}{2}$ or $\frac{1}{4}$ of the model's frame size) to produce an output at a shorter time. Of course, this requires a model which is capable of processing the input within this shorter time period.

Since considerable resources are required for the training of such systems, the training procedure was performed on a powerful GPU machine. Next, the pre-trained AECNN speech enhancement models were implemented in the developed framework and executed in real-time on the CPU of the Raspberry Pi to perform real-time speech enhancement on the audio input.

3 EVALUATION

We used the dataset presented by Valentini et al. [14] to evaluate and compare the various systems. The database is derived from the Voice Bank corpus [15], from which recordings from 28 speakers were chosen for the training set (11572 utterances) and 2 for the test set (824 utterances). The training set simulates 40 different noisy scenarios with 10 different noise conditions (2 artificial and 8 from the DEMAND database [16]) at signal-to-noise ratios (SNRs) of 0, 5, 10 and 15 dB. The test set was created using 5 noise conditions from the DEMAND database (different from the training noise conditions) added at SNRs 2.5, 7.5, 12.5 and 17.5 dB. In our case, all the utterances were downsampled from 48 kHz to 16 kHz. The models were trained using the Adam optimizer [17] for 40 epochs with a learning rate of 0.0002, using a batch-size of 100. The L1 distance between the generated samples $\hat{x}$ and the clean examples x was used as the penalty, i.e. $L_{1loss} = \sum_{i=1}^{n} ||\hat{x} - x||_1$. The speech signals were windowed using sliding windows of various lengths with 50% overlap.

Frontend comparison: The models were developed based on [18] and implemented in Keras [19] using Tensorflow [20] as the backend. For each trained layout, we evaluated the speech enhancement performance and execution time on the portable platform using Keras as the frontend. Using Tensorflow, these Keras implementations were then converted in a Protocol Buffer (protobuf) representation and the corresponding graphs, together with their respective weights, were saved in a binary format (.pb). The converted models were afterwards evaluated using Tensorflow as the frontend. This step tests whether the type of frontend impacts the execution time.

Speech enhancement evaluation: The speech enhancement performance was evaluated using the perceptual evaluation of speech quality metric (PESQ) [21], measured in terms of mean opinion score (MOS), and the segmental SNR (segSNR). Higher values of these metrics indicate better performance. For the test set used, the enhanced signals were evaluated for each different combination of overlap and frame buffering supported by our framework (namely 0% and 50% overlap as well as 0%, 50% and 75% frame buffering).

Execution time: We measured the execution times of the various layouts on the Raspberry Pi for different frame buffering and overlapping percentages. The execution time corresponds to the time required for the inference of each model as well the pre-processing of the input frame and the post-processing of the output frame, respectively. This includes the necessary operations for the required buffering of frames and implementation of overlap. As in the case of speech enhancement evaluation, we measured the execution time for each different combination of overlap and frame buffering (i.e. 0% and 50% overlap and 0%, 50% and 75% frame buffering). The total processing time was measured for each frame on the Raspberry Pi in terms of milliseconds. For statistical validation the reported time corresponds to the average over 10 runs.

Computational complexity: We measure the computational cost of each layout using the total amount of floating-point operations (FLOPs) and trainable parameters. The complexity of each architecture was computed using Tensorflow.

Architectures: Four input / output layer sizes L were used: 1024, 512, 256 and 128 samples, corresponding to 64, 32, 16 and 8 ms resolution respectively for a sampling rate of 16 kHz. For each layer size, different combinations of kernel widths and filter lengths were used, resulting in varying amounts of trainable

[1] https://github.com/jackaudio/jack2
[2] https://github.com/spatialaudio/jackclient-python
[3] https://github.com/HearingTechnology/aecnn-rpi

parameters. Several activations were also tested for each layer (i.e. tanh, ReLU, LeakyReLU [22], PReLU [23], ELU [24]) and were compared in terms of speech enhancement performance and inference time. Additionally, models were trained both with and without the inclusion of bias in the convolutional layers. The architectures and the parameters of each trained model used are shown in Table 1. For simplicity, only the encoder filter sizes are shown, but the mirrored version of the kernel is used for the shaping of the decoder, resulting in twice the amount of layers. Starting with a number of samples L, the signal size L_i gets halved in each consecutive layer of the encoder.

Table 1. Used AECNN architectures. L corresponds to the input/output frame size of each architecture in samples, the *kernel* contains the number of filters k_i (feature maps) applied in each convolutional layer of the encoder and w is the filter width used in each case. The use (or not) of *bias* is also denoted in the table with a 1 or a 0 respectively. The total number of *parameters* is shown for each model and the number of million floating-point operations *(G-FLOPs)*.

Name	L	Kernel	w	Bias	Activation	Parameters	G-FLOPs
$AECNN_{1024,1}$	1024	[128,64,64,32,32,16,16]	31	1	PReLU	3,876,193	19.38
$AECNN_{1024,2}$	1024	[128,64,64,32,32,16,16]	15	1	PReLU	1,944,673	9.72
$AECNN_{1024,3}$	1024	[128,64,64,32,32,16,16]	15	0	PReLU	1,942,928	9.71
$AECNN_{1024,4}$	1024	[128,64,64,32,32,16,16]	15	0	LeakyReLU	1,810,819	9.05
$AECNN_{1024,5}$	1024	[64,32,16,16,8]	15	0	LeakyReLU	306,375	1.53
$AECNN_{512,1}$	512	[64,32,32,32,16,16]	15	0	LeakyReLU	566,657	2.83
$AECNN_{512,2}$	512	[64,32,32,16,16,16]	15	0	PReLU	516,748	2.58
$AECNN_{512,3}$	512	[64,32,32,16,16,8]	15	1	PReLU	500,347	2.50
$AECNN_{512,4}$	512	[64,32,32,16,16,8]	15	0	PReLU	486,100	2.43
$AECNN_{512,5}$	512	[64,32,32,16,16,8]	11	0	PReLU	367,540	1.84
$AECNN_{512,6}$	512	[32,16,16,8,8]	15	0	PReLU	136,848	0.68
$AECNN_{256,1}$	256	[32,32,16,16,16]	15	0	PReLU	257,928	1.29
$AECNN_{256,2}$	256	[32,32,16,16,16]	15	0	LeakyReLU	232,575	1.16
$AECNN_{256,3}$	256	[32,32,16,16,16]	15	0	ELU	232,575	1.16
$AECNN_{256,4}$	256	[32,32,16,16,16]	15	0	ReLU	232,575	1.16
$AECNN_{256,5}$	256	[32,32,16,16,16]	15	0	tanh	232,575	1.16
$AECNN_{256,6}$	256	[32,32,16,16,16]	11	0	PReLU	195,912	0.98
$AECNN_{256,7}$	256	[32,32,16,16,8]	21	1	LeakyReLU	300,436	1.50
$AECNN_{256,8}$	256	[32,32,16,16,8]	15	1	LeakyReLU	214,756	1.07
$AECNN_{256,9}$	256	[32,32,16,16,8]	15	0	PReLU	233,424	1.17
$AECNN_{256,10}$	256	[32,32,16,16,8]	15	0	LeakyReLU	214,215	1.07
$AECNN_{128,1}$	128	[32,32,16,16,16]	21	1	LeakyReLU	326,188	1.63
$AECNN_{128,2}$	128	[32,32,16,16,8]	15	1	LeakyReLU	214,756	1.07
$AECNN_{128,3}$	128	[32,32,16,16,8]	15	0	ELU	214,215	1.07
$AECNN_{128,4}$	128	[32,16,16,8]	15	0	PReLU	107,468	0.54

No pre-emphasis filter was applied to the input samples, because the de-emphasis filtering was found to significantly increase the execution time on the Raspberry Pi. On the other hand, we trained some models using a [-0.95, 1] impulse response pre-emphasis filter, which was applied to the frames of the training set data. This way, we could test whether efficient noise suppression would be performed without applying pre-emphasis to the audio input of the Raspberry Pi (and de-emphasis respectively), considering the fact that a pre-emphasis filtering is sometimes introduced by the DAC of the hardware (e.g. soundcard). However, the performance of the models without pre-emphasis filtering was poor and therefore is not reported here.

Floating-point precision: Single-precision floating-point format (32-bit) was used both for the audio server and for the speech enhancement models. Lowering the floating-point precision (e.g. 16-bit) was not found to decrease the execution time. Instead, more time was required to execute the same architecture on the CPU of the Raspberry Pi with decreased precision. The reason is that 16-bit floating-point numbers are not supported by CPUs directly. Therefore, these operations need to be emulated, which causes this comparatively slow performance. On the other hand, GPUs can support half-precision arithmetic natively and such conversion can significantly decrease the execution time. Since all the inference is performed on the CPU of the Raspberry Pi, there was no benefit in lowering the floating-point format.

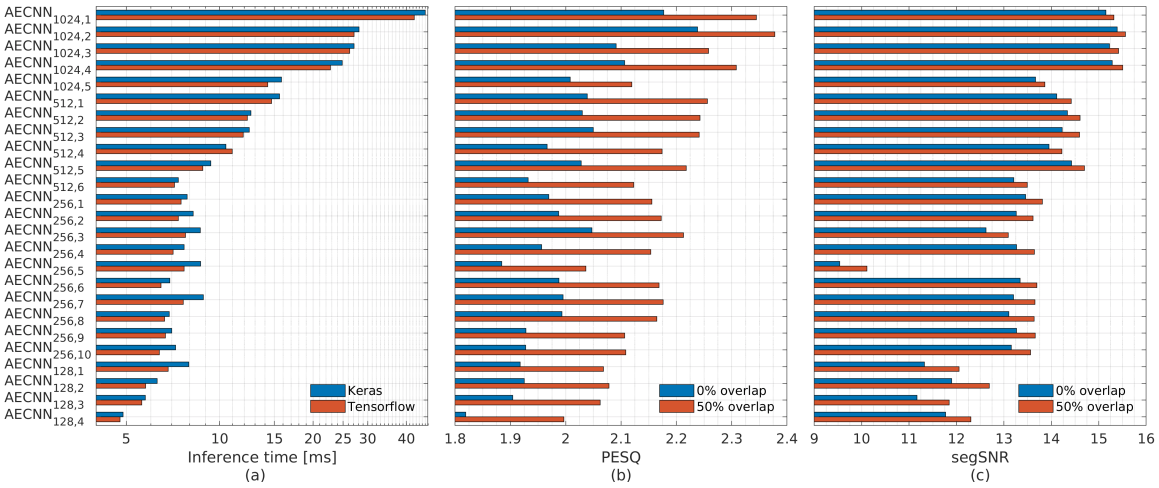

Figure 2. Achieved execution time for each frontend (a) and speech enhancement performance for each model with and without overlap between frames (b,c). Higher values of the objective metrics PESQ and segSNR indicate better performance. The results shown are without the use of frame buffering.

4 RESULTS

The results for each trained model in terms of execution time and speech enhancement performance are shown in Figure 2.

Frontend comparison: Figure 2(a) shows that almost every model is executed faster using Tensorflow as the frontend. Tensorflow inference was found to be about 6.5% faster than Keras on average, providing up to 4 ms improvement in large models. The speech enhancement performance of the Keras and Tensorflow models was almost identical and is not reported here.

Model parameters: The most efficient activation layers were found to be ELU, PReLU and LeakyReLU, both in terms of performance and in terms of execution time. Considering speech enhancement performance, ELU was the best choice, but it was rather slow compared to PReLU and LeakyReLU. PReLU and LeakyReLU yielded similar noise suppression performance, but the former introduced more parameters and floating-points operations compared to the rest of the activation layers. This made PReLU a bit slower, especially for large models and Keras implementations, but its use speeded up the execution of small models in Keras.

Apart from tanh and ReLU, the rest of the activation layers were comparable to each other with subtle differences between them. Considering both factors, PReLU and LeakyReLU were found to give the best trade-off between evaluated performance and execution time, with LeakyReLU being clearly the best choice for models with more than 500,000 trainable parameters and PReLU being the best choice for small-sized models implemented in Keras.

The inclusion of bias in the convolutional layers was found to significantly increase the inference times of the models, introducing more trainable parameters and operations (FLOPs). A performance improvement was apparent, but bias was not found to be helpful enough for small-sized AECNN models. Regarding the selection of the filter width w, a length of 31 is generally preferred for convolutional speech enhancement models [13, 18]. In our case, it resulted in significantly larger and slower models, without providing any remarkable improvement in terms of speech enhancement performance. Instead, setting the filter length to 15, resulted in faster models with similar (sometimes even better) performance. Lowering the filter length even further (for instance $w = 11$) was found to be beneficial in some cases, especially for small-sized models, while increasing the filter length above 15 (e.g. 21) did not yield a substantial improvement.

Overlap: Figure 2 shows that higher performance is always achieved with the introduction of overlap between frames, especially in the case of a PESQ evaluation (Fig. 2(b)). Although overlap systematically improves the noise suppression performance, it also requires that the processing of each frame is performed within half of the initial time. This is a significant sacrifice to make but, in most cases, the use of a smaller-sized model with overlapping windows resulted in a better performance compared to a model with more layers and no overlap.

Performance vs learning power: DNN-based models are known to be inefficient in utilizing their full learn-

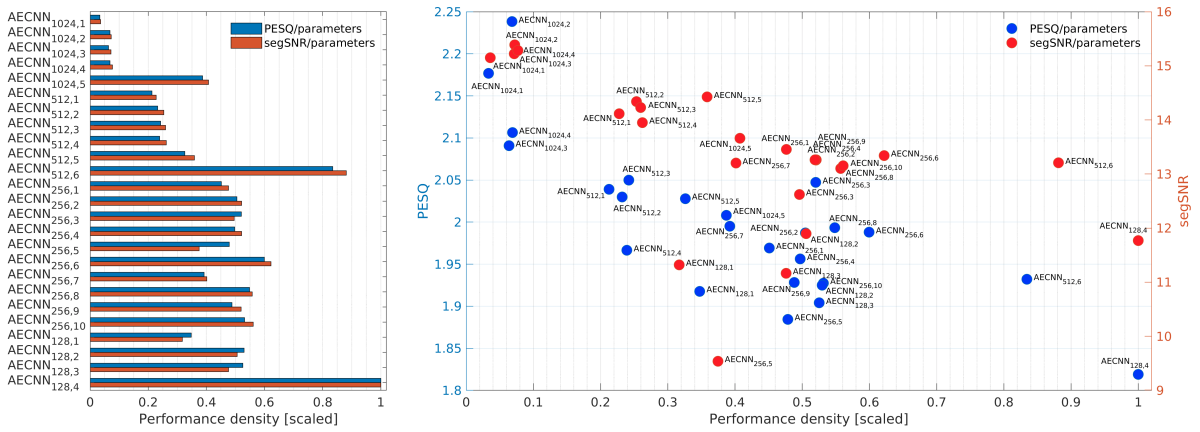

Figure 3. **Left:** Performance density, **Right:** Performance vs performance density (in terms of PESQ and segSNR). The performance density indicates how efficiently each model uses its parameters.

ing power, considered as the number of parameters with respect to the degrees of freedom. Inspired by [25], we measured the performance density of each model, i.e. the performance of each architecture (in terms of PESQ and segSNR) divided by the number of trainable parameters. The higher this value, the higher the efficiency of the respective architecture. To compare these two performance ratios (PESQ/params & segSNR/params) across the different architectures, we scaled each by its maximum (absolute) value. Figure 3(a) shows that the models that use their parameters most efficiently are $AECNN_{512,6}$, $AECNN_{256,6}$ and $AECNN_{128,4}$.

To evaluate the efficiency, we plot the performance density against the two performance metrics PESQ and segSNR, which permits us to determine the desired trade-off. Figure 3(b) shows the $AECNN_{256,6}$ is among the most efficient models, while providing a high PESQ (left axis) as well as a high segSNR value (right axis). Among the models with the highest performance, we can see that the $AECNN_{512,3}$ and $AECNN_{512,5}$ architectures use their parameters more efficiently. On the other hand, models such as $AECNN_{256,5}$ failed to make good use of their parameters while yielding a poor speech enhancement performance.

Latency comparison: Using our Python framework, we compared the AECNN models in real-time for three different processing latencies: 32, 16 and 8 ms. For a sampling rate of 16 kHz, this corresponds to frame sizes of 1024, 512 and 256 samples for the audio server (JACK). Based on their execution times, the models that could cope with each required latency limitation were chosen. Table 2 shows the best AECNN model setups for each case, based on their speech enhancement performance.

5 CONCLUSIONS

We introduced a real-time framework in Python, able to impelement DNN-based speech enhancement models in real-time. Using this framework, we executed auto-encoder models on a low-resource platform to provide an enhanced output with low latency. An important question raised throughout the study was the amount of temporal context necessary to achieve efficient speech enhancement. Table 2 shows that, for low latencies, models with an input of 256 samples provided the best performance. Although the temporal context decreases, these models performed better than the small-sized models with 512 samples input. The use of a smaller input seems to be beneficial in this case, providing more parameters to be trained. However, a resolution of 128 samples for the input layer did not yield the same result, since such models achieved much worse performance than what was achieved by models with larger inputs (Table 2, Figure 2). This shows that efficient speech enhancement can be performed with less than 32ms temporal context (in our case 16 ms), but a drop of performance is apparent for a resolution of 8 ms.

Different architectures were necessary to achieve the desired processing latency in each case (Table 2). Using our real-time framework, we found that the limitations in terms of number of parameters and kernel sizes for our AECNN models were: up to 7.5 million parameters and 14 layers for 64 ms latency, up to 3 million parameters and 14 layers for 32 ms latency, up to 900 thousand parameters and 12 layers for 16 ms latency and up to 350 thousand parameters with 10 layers to achieve 8 ms latency. It should be mentioned here that the theoretical latency of the processing algorithm is considered throughout this paper. This was done to provide a lower limit for which the proposed speech enhancement models can be implemented. However,

Table 2. The most efficient speech enhancement setups for three different latencies: 32, 16 and 8 ms. For each model, we show the required buffering as a percentage of the original frame, the percentage of overlap applied (if any), the resulting execution time and the speech enhancement performance results. The best results obtained for each latency are highlighted in bold font.

Name	Parameters	Buffer	Overlap	Execution time	PESQ	segSNR
Unprocessed					1.9702	8.7735
64 ms latency						
$AECNN_{1024,1}$	3,876,193	0%	0%	42.3552	2.1769	15.1496
$AECNN_{1024,2}$	1,944,673	0%	50%	27.4364	**2.3778**	**15.5603**
32 ms latency						
$AECNN_{1024,2}$	1,944,673	50%	0%	27.3441	2.2549	**15.2189**
$AECNN_{512,1}$	566,657	0%	50%	14.8914	**2.2561**	14.4206
$AECNN_{512,3}$	500,347	0%	50%	12.0943	2.241	14.5937
16 ms latency						
$AECNN_{1024,5}$	306,375	75%	0%	14.2758	1.9977	12.931
$AECNN_{512,3}$	500,347	50%	0%	11.9878	2.138	14.197
$AECNN_{512,5}$	367,540	50%	0%	8.9128	2.0846	**14.3374**
$AECNN_{256,1}$	257,928	0%	50%	7.7154	2.1559	13.8087
$AECNN_{256,3}$	232,575	0%	50%	7.9564	**2.2127**	13.0943
8 ms latency						
$AECNN_{512,6}$	136,848	75%	0%	7.2789	1.9513	12.3311
$AECNN_{256,1}$	257,928	50%	0%	7.6383	2.0296	**13.3161**
$AECNN_{256,3}$	232,575	50%	0%	7.9285	**2.0953**	12.4309
$AECNN_{128,2}$	214,756	0%	0%	5.7714	1.9251	11.8959

the experienced (measured) latency is always higher than the theoretical one and depends on several factors such as the audio server backend or the analog-digital and digital-analog converters of the sound card.

During our evaluation, we observed there was a high variability in the values of the execution time on the Raspberry Pi. This depends on the computational load at a given time, where small variances can make a big difference, as well as on the temperature of the hardware. We must also take into consideration the latency added up by our Python framework, which was measured to be about 0.18 ms. In any case, a safe approach is to work with models that do not easily reach the latency limitation in terms of execution time.

Lastly, audio drop-outs can easily occur in Python due to its global interpreter lock (GIL) or its garbage collector. Because of this, Python is not the best choice for real-time processing. While being aware of this limitation, we nevertheless used Python because of its ease of implementation and customisation for DNN systems. We did not focus on achieving the fastest possible execution but rather aimed to develop an easy-to-use framework where DNN-based speech enhancement methods can be implemented on a low-resource platform and tested in real-time. To design faster and more reliable real-time audio processing applications, the audio process function as well as the speech enhancement algorithm should be re-implemented in a different programming language such as C or C++.

ACKNOWLEDGEMENTS

Work supported by European Research Council starting grant ERC-StG-678120 (RobSpear).

REFERENCES

[1] Ortega-Garcia J, Gonzalez-Rodriguez J. Overview of speech enhancement techniques for automatic speaker recognition. In 1996. p. 929–32 vol.2.

[2] Maas A, Le QV, O'Neil TM, Vinyals O, Nguyen P, Ng AY. Recurrent Neural Networks for Noise Reduction in Robust ASR. In: INTERSPEECH. 2012.

[3] Yang L-P, Fu Q-J. Spectral subtraction-based speech enhancement for cochlear implant patients in background noise (L). The Journal of the Acoustical Society of America. 2005 Apr 1;117:1001–4.

[4] Loizou PC. Speech Enhancement: Theory and Practice. 2nd ed. Boca Raton, FL, USA: CRC Press, Inc.; 2013.

[5] F. Boll S. Suppression of Acoustic Noise in Speech Using Spectral Subtraction. Acoustics, Speech and Signal Processing, IEEE Transactions on. 1979 May 1;27:113–20.

[6] Berouti M, Schwartz R, Makhoul J. Enhancement of speech corrupted by acoustic noise. In: ICASSP '79 IEEE International Conference on Acoustics, Speech, and Signal Processing. 1979. p. 208–11.

[7] Scalart P, Filho JV. Speech Enhancement Based on a Priori Signal to Noise Estimation. In: Proceedings of the Acoustics, Speech, and Signal Processing, 1996 On Conference Proceedings, 1996 IEEE International Conference - Volume 02 . Washington, DC, USA: IEEE Computer Society; 1996. p. 629–632. (ICASSP '96).

[8] Tsoukalas DE, Mourjopoulos JN, Kokkinakis G. Speech enhancement based on audible noise suppression. IEEE Transactions on Speech and Audio Processing. 1997 Nov;5(6):497–514.

[9] Narayanan A, Wang D. Ideal Ratio Mask Estimation Using Deep Neural Networks for Robust Speech Recognition. In 2013. p. 7092–6.

[10] Wang Y, Narayanan A, Wang D. On Training Targets for Supervised Speech Separation. IEEE/ACM Trans Audio Speech Lang Process. 2014 Dec;22(12):1849–58.

[11] Paliwal K, Wójcicki K, Shannon B. The Importance of Phase in Speech Enhancement. Speech Commun. 2011 Apr;53(4):465–494.

[12] Lu X, Tsao Y, Matsuda S, Hori C. Speech enhancement based on deep denoising autoencoder. In: INTERSPEECH. 2013.

[13] Pascual S, Bonafonte A, Serrà J. SEGAN: Speech Enhancement Generative Adversarial Network. In 2017. p. 3642–6.

[14] Botinhao CV, Wang X, Takaki S, Yamagishi J. Investigating RNN-based speech enhancement methods for noise-robust Text-to-Speech. In: Proceedings of 9th ISCA Speech Synthesis Workshop. 2016. p. 159–65.

[15] Veaux C, Yamagishi J, King S. The voice bank corpus: Design, collection and data analysis of a large regional accent speech database. Oriental COCOSDA held jointly with 2013 Conference on Asian Spoken Language Research and Evaluation (O-COCOSDA/CASLRE), 2013 International Conference. 2013 Nov.

[16] Thiemann J, Ito N, Vincent E. The Diverse Environments Multi-channel Acoustic Noise Database (DEMAND): A database of multichannel environmental noise recordings. In 2013.

[17] Kingma D, Ba J. Adam: A Method for Stochastic Optimization. International Conference on Learning Representations. 2014 Dec 22.

[18] Baby D, Verhulst S. SERGAN: Speech enhancement using relativistic generative adversarial networks with gradient penalty, In: IEEE-ICASSP, Brighton, UK. May 2019

[19] Chollet F, et al. Keras. https://keras.io, 2015.

[20] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. 2016 Mar 14.

[21] Rix AW, Beerends JG, Hollier MP, Hekstra AP. Perceptual Evaluation of Speech Quality (PESQ)-a New Method for Speech Quality Assessment of Telephone Networks and Codecs. In: Proceedings of the Acoustics, Speech, and Signal Processing, 200 On IEEE International Conference - Volume 02. Washington, DC, USA: IEEE Computer Society; 2001. p. 749–752. (ICASSP '01).

[22] Maas A, Hannun A, Ng A. Rectifier nonlinearities improve neural network acoustic models. In: International Conference on Machine Learning (ICML). 2013.

[23] He K, Zhang X, Ren S, Sun J. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. IEEE International Conference on Computer Vision (ICCV 2015). 2015 Feb 6;1502.

[24] Clevert D-A, Unterthiner T, Hochreiter S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). Under Review of ICLR2016 (1997). 2015 Nov 23;

[25] Canziani A, Paszke A, Culurciello E. An Analysis of Deep Neural Network Models for Practical Applications. 2016 May 24;