FAST TEMPORAL CONVOLUTIONS FOR REAL-TIME AUDIO SIGNAL PROCESSING

Stepan Miklanek and Jiri Schimmel

Department of Telecommunications
Brno University of Technology, Brno, Czech Republic
`stepan.miklanek@vut.cz | schimmel@vut.cz`

ABSTRACT

This paper introduces the possibilities of optimizing neural network convolutional layers for modeling nonlinear audio systems and effects. Enhanced methods for real-time dilated convolutions are presented to achieve faster signal processing times than in previous work. Due to the improved implementation of convolutional layers, a significant decrease in computational requirements was observed and validated on different configurations of single layers with dilated convolutions and WaveNet-style feedforward neural network models. In most cases, equivalent signal processing times were achieved to those using recurrent neural networks with Long Short-Term Memory units and Gated Recurrent Units, which are considered state-of-the-art in the field of black-box virtual analog modeling.

1. INTRODUCTION

Recently, we have seen the adoption of deep learning methods in all approaches to Virtual Analog (VA) modeling of nonlinear systems [1, 2], guitar amplifiers [3, 4, 5, 6], time-varying effects [7, 8, 9], and other digital audio effects [10, 11]. In addition to VA modeling, deep learning methods have also found applications in digital filter design [12] and target filter frequency response matching [13].

The use of neural networks dominates mainly within the so-called "black-box" approach, where knowledge of the internal layout of the modeled systems is not required [14]. In this case, these methods are a very efficient way to capture the characteristics of a given device simply by obtaining a dataset of input and output signals and then training a neural network that can emulate the properties of any system regardless of its complexity. However, analog units usually have user controls to change their behavior, so it is usually necessary to acquire input and output signals with different device settings to create models with variable parameters. Creating such a dataset could be time-consuming, but neural networks can extrapolate between discrete values of captured control settings. It has been shown that neural models, once trained, can very faithfully simulate control positions that were not used for training with marginal error [5, 15].

Deep learning methods have also found their application in "white-box" VA modeling, which is a process of creating digital versions of analog circuits with the possibility of changing the values of virtual components [16]. In [17], it was demonstrated that it is possible to fine-tune the initial values of VA circuit components using a procedure where a discrete circuit model with differentiable scaling coefficients is created.

In 2016, we saw the advent of the WaveNet convolutional neural network architecture, which revolutionized the area of human speech synthesis [18]. Variations of this structure were derived in later years and successfully used to model distortion pedals [15], tube guitar amplifiers [19], and effects such as compressors or reverbs [20, 21]. These types of neural networks became also known as Temporal Convolutional Networks (TCNs). Besides to their application in digital audio effects, they have recently been utilized in tasks concerned with analyzing the rhythmic structure of songs [22] or sound event localization [23]. Both the original WaveNet model and different TCN variations consist of stacks of multichannel convolutional layers with dilated kernels, which help to store information about previous development of time sequences. This property is especially useful when working with stateful systems because their behavior is typically dependent on past values of digital signal samples [24].

The TCNs are, at first sight, an ideal candidate for modeling nonlinear systems and digital effects. However, previous work has shown that using a large number of convolutional layers is necessary to obtain faithful models [6]. Thus, mainly structures based on Recurrent Neural Networks (RNNs) have lately been investigated for nonlinear system and VA modeling [6, 7, 25]. The main reason is the reduced complexity of these structures and the resulting lower computational demands. Another reason for examining RNNs is their close relationship to previous modeling techniques based on a nonlinear state space representation [24, 25].

RNNs seem to be the most suitable method for black-box modeling nonlinear systems due to their processing efficiency and relatively low complexity [5]. However, we would like to revisit previously proposed TCN models [15] and explore whether they can approach RNN-like processing speeds thanks to optimized convolutional layers. In later sections of the paper, we verify the computational speed-up on different configurations of single convolutional layers and neural networks.

The rest of this article is organized as follows. Section 2 describes temporal convolutions used in neural networks and their realization for real-time audio signal processing. This section also describes our convolution algorithm. Section 3 summarizes TCNs, which were previously used to model nonlinear systems. Section 4 validates the proposed implementation of convolutional layers by measuring the signal processing speed compared to implementations presented in previous work. Finally, Section 5 concludes the paper.

2. TEMPORAL CONVOLUTIONS

First of all, we need to establish mathematical definitions of convolution operations suitable for real-time signal processing in digital

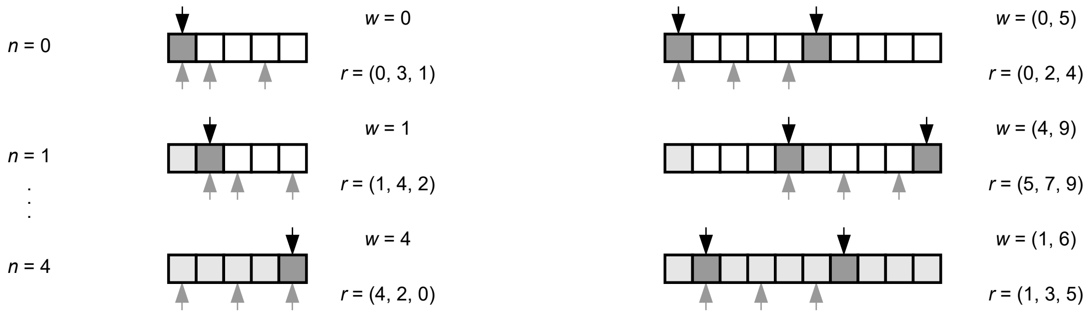

Figure 1: *Comparison between single buffer (left) and double buffered state (right) in terms of write and read pointer movement in subsequent time steps n for convolutional layer with kernel of length 3 and dilation rate of 2. The black arrows indicate the write pointers, and the grey arrows indicate the read pointers.*

audio effects. It is usually desirable to work with either the current signal sample or possibly some number of previous samples. The main reason is that we do not have future signal values when processing the signal in real time. Therefore, in this paper, we solely consider discrete causal convolutions, which depend only on the current and preceding samples of the signal.

The discrete causal convolution of input signal x and convolution kernel h is defined by the equation

$$(x * h)[n] = \sum_{m=0}^{M-1} x[n-m]h[m], \tag{1}$$

where m is the coefficient index of kernel h, M is the length of kernel h, and n is the discrete time index. Thus, it is necessary to have $M-1$ previous samples of signal x stored in memory when implementing this operation. TCN convolutional layers also contain a dilation rate parameter that causes the kernels to stretch back in time. Thus, we can can modify (1) as

$$(x * h)[n] = \sum_{m=0}^{M-1} x[n-md]h[m], \tag{2}$$

where d is the dilation rate. This modification necessitates increasing the required buffer size, which can be calculated according to the equation

$$s_{\text{buffer}} = d(M-1)+1. \tag{3}$$

2.1. Buffer Variants

There are two basic strategies for implementing buffers for discrete causal convolutions for real-time signal processing. The first is using a single buffer whose size is computed by (3). The second approach uses a double buffer, which is twice as long. Besides memory size, the main difference between these methods is the movement of the write and read pointers during each time step. This is shown in Figure 1.

Both approaches achieve the same result, but the single buffer requires only one memory write per time step. This feature is advantageous in terms of processing time since multi-channel convolutions require storing a vector of input signals instead of a single sample. In our case, we chose the single buffer method because we strive to achieve the lowest possible computational complexity.

2.2. Proposed Convolution Algorithm

So far, there are only two (available) implementations of neural convolutional layers for real-time audio signal processing. The first one uses previously described single buffering [26], and the later one utilizes double buffering [27].

Algorithm 1 Convolutional Layer Forward Propagation

Variables: Input channels x, output channels $\hat{y}$, layer output size s_{out}, write pointer w, vector of read pointers r, kernel size M, container of kernel weight arrays H, bias vector b, buffer array B, buffer size s_{buffer}, array of selected buffer columns B_{cols}

```
 1: function FORWARD($x, \hat{y}$)
 2:     $B[all, w] \leftarrow x$
 3:     Set the vector of read pointers $r$
 4:     for each $i < K$ do
 5:         $B_{\text{cols}}[all, i] \leftarrow B[all, r(i)]$
 6:     for each $i < s_{\text{out}}$ do
 7:         $\hat{y}(i) \leftarrow sum(B_{\text{cols}} \odot H(i)) + b(i)$
 8:     if $w < (s_{\text{buffer}} - 1)$ then
 9:         $w \leftarrow w + 1$
10:     else
11:         $w \leftarrow 0$
```

Algorithm 1 describes our approach to forward propagation through the convolutional layer. The input of the convolutional layer is a vector x, where each element corresponds to one sample of a given audio channel. This vector is then inserted into the two-dimensional buffer array's column, whose index is determined by the write pointer w.

Then it is necessary to select the memory array columns whose indices correspond to the elements of the vector of read pointers r. The selected columns of buffer B are stored in array B_{cols}, whose number of rows is equal to the number of input channels, and the number of columns equals the kernel length M. The convolution itself is then performed as the sum of the Hadamard product of B_{cols} with ith element of the kernel array container H.

Each array from container H contains a number of convolution kernels equal to the count of input channels and must also have the same dimensions as the B_{cols} array. The number of convolution operations is directly affected by the required number of output channels s_{out}. The result of the convolution operations is

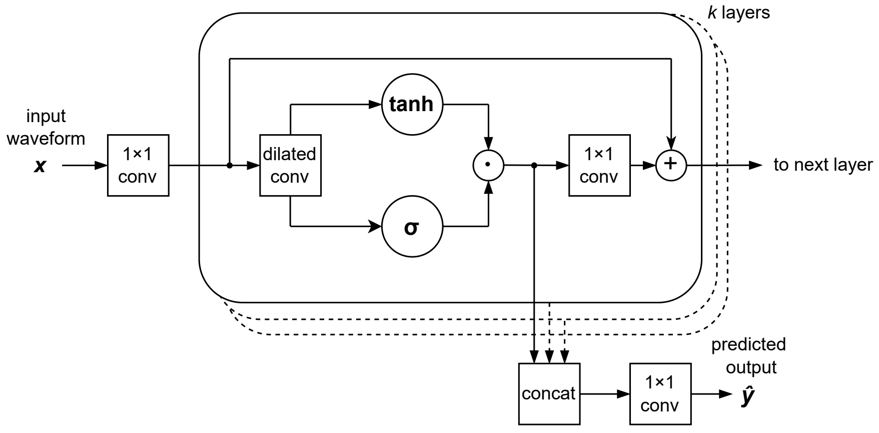

Figure 2: *Block diagram of TCN (WaveNet-style) model with gated activation.*

then assigned into the output vector $\hat{y}$. The only difference from the discrete causal convolution definition is the addition of a bias b, a vector of offset constants added to the output $\hat{y}$.[1]

The proposed algorithm was implemented using the Eigen library for C++. The Eigen library allows using fixed-sized variables, which can significantly speed up calculations, especially when using small matrices and vectors [28]. This has already been leveraged in the RTNeural framework for real-time neural network inferencing [27].

An obvious disadvantage of using fixed-sized variables for neural network models is the impossibility of changing the structure of the model during the run-time of a standalone program or an audio plugin. Nevertheless, this approach may be suitable for black-box modeling, as there is usually no need to change the neural network structure on the fly. In this work, we experimented with two versions of a convolutional layer, the first with dynamic variables and the second with static variables initialized at compile time.

3. NEURAL NETWORK MODELS

From previous research, we are familiar with techniques for identification of nonlinear systems that follow the structure of a Wiener or Wiener-Hammerstein model [14, 29]. These models are composed of linear filters and a static nonlinear transfer functions, which is also the case for TCNs.

To validate the proposed convolution algorithm, we chose the neural network model presented in [19]. We decided to choose this model mainly because of the availability of source code and reference signal processing times from previous work.

This model is a feedforward variant of the original WaveNet neural network [18] and consists of the following blocks. The first block is a multi-channel 1×1 convolution with kernels of size 1, whose input is a raw audio waveform. This step can be described as a single-input multiple-output (SIMO) convolution. In [6], the authors do not describe this block, but the implementation they refer to includes this convolution operation. For this reason, we decided to incorporate it as well. However, it is likely that this convolutional layer is redundant.

[1]Biases are additional trainable parameters of neural networks.

The first block is followed by a stack of k convolutional layers as shown in Figure 2. These layers have the same internal structure except for the dilation rate parameter d_k of the dilated convolution. The output z_k of the dilated convolution at the beginning of each layer can be expressed as

$$z_k[n] = f[(H_k * x_k)[n] + b_k], \tag{4}$$

where H_k is the dilated causal convolution kernel, x_k is the layer input, b_k is the bias term, and $f(\cdot)$ is the nonlinear activation function. The convolutional layers in this model have multiple channels, so the filtering is done as a multiple-input multiple-output (MIMO) convolution with the kernel H_k. The individual filters in this kernel have impulse responses

$$h[n] = \sum_{m=0}^{M-1} w_m \delta[n - md_k], \tag{5}$$

where $\delta[n]$ is the Kronecker delta function, and w_m are the non-zero coefficients of the filters learned by the network.

The individual layers also include residual connections, so the input to the next layer is described by

$$x_{k+1}[n] = W_k z_k[n] + x_k[n], \tag{6}$$

where W_k is a 1×1 convolution kernel that is responsible for the ratio between the layer input x_k and the layer output z_k before the next layer. Finally, outputs z_k from each layer are fed into the 1×1 convolution block, which has only one output channel and mixes all the outputs linearly.

The authors of [5, 6, 15, 19] performed a detailed investigation of different configurations of this model in terms of the choice of nonlinear functions, the number of layers, and the number of convolution channels. For the sake of brevity, we decided to select only the variant a gated activation function, so (4) can be modified to

$$z_k[n] = \tanh[(H_{fk} * x_k)[n] + b_{fk}] \odot \sigma[(H_{gk} * x_k)[n] + b_{gk}], \tag{7}$$

where $\tanh$ is the hyperbolic tangent, σ is the logistic sigmoid function, symbol $\odot$ is the Hadamard product, H_{fk} is the filter kernel, H_{gk} is the gate kernel, b_{fk} and b_{gk} are the filter and gate biases, respectively.

4. RESULTS

The evaluation process of signal processing speed using our algorithm was performed first on individual convolutional layers with different parameters and then on TCN models with different numbers of layers. We included the reference WaveNetVA implementation from [26] and the one from the RTNeural framework [27]. We measured how long it takes to process 1 second of a signal at a sampling rate of 44.1 kHz in all tests. If the signal took longer than 1 second to process, the layer or model could not be run in real time. We repeated all measurements ten times. Therefore, all values in the following figures are average signal processing times. The processing speed is expressed as a factor of the requirement for real-time application.

Although signal processing times of TCNs are reported in previous work [5, 6], to fairly compare the WaveNetVA implementation with ours, we have re-measured all processing times. The results could be different, in particular, due to running the tests on a different computer. As part of this work, we also compare the signal processing speed of TCNs versus RNNs. To perform this comparison, we implemented our version of RNNs because the implementation's source code from previous literature is not freely available. In addition, we use the RNN signal processing times from [5] as a reference.

4.1. Single Layer Performance

The performance of single convolutional layers was evaluated in terms of dilation rate, kernel size, and the number of convolution channels. The RTNeural framework offers the use of layers with fixed-sized and dynamic variables. Therefore, we chose layers with fixed-sized variables in the case of the RTNeural library due to less computational overhead.

Figure 3 depicts the dependency of increasing dilation rate and the consequent effect on processing speed. This test used a layer with only one convolution channel and kernel size of $M = 3$. Ideally, increasing the dilation rate should not result in a decrease in processing speed. In the case of both our implementations and WaveNetVA, the processing speed was almost constant at all dilation rates. However, with RTNeural, we observed that the signal processing time was slower with an increasing dilation rate. This is because RTNeural performs the dilation rate increase by inserting zero values between the non-zero filter coefficients. Effectively, the convolution filters are extended, and the computational intensity grows. Although it would be possible to modify the RTNeural convolutional layer so that this slowdown does not occur, we have not addressed this in our work.

Due to these findings, we decided to exclude RTNeural from the tests in Section 4.2 because TCN models use layers with large dilation rates, and thus their performance would be quite slow.

In Figure 4, it is possible to observe the dependence of computing time on the kernel size for a layer with a single convolution channel. In this test, the RTNeural library achieved the shortest signal processing times in all cases except one. With longer filters, the processing speed decreases greatly for all implementations. TCNs usually do not need long convolutional filters, and therefore this comparison is rather illustrative.

Our static implementation is slightly slower than RTNeural in terms of kernel length because we use a variable with a dynamic size for the convolutional layer memory. When using large dilation rates, the memory array size requirements increase. Hence, it is not appropriate to use a fixed-sized variable as stated in the Eigen library documentation [28].

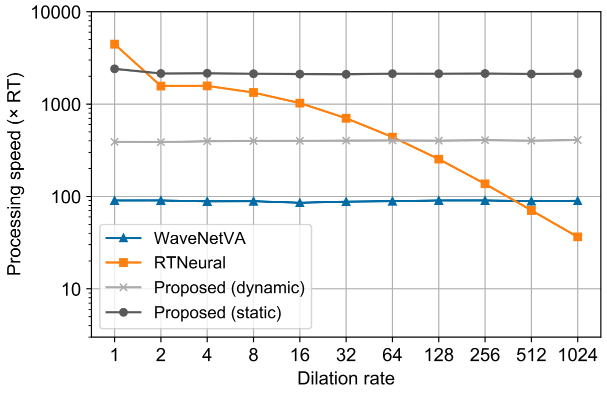

Figure 3: *The processing speed of a single convolutional layer with different dilation rates.*

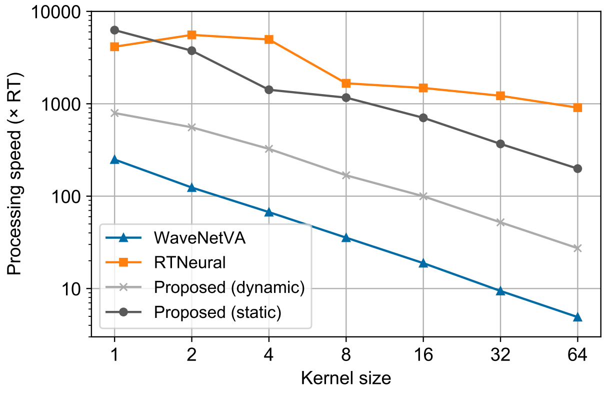

Figure 4: *The processing speed of a single convolutional layer with different kernel sizes.*

Figure 5 shows the processing speed of a layer with a different number of convolution channels, a kernel size of $M = 1$ and a dilation factor of $d = 1$. With a lower convolution channel count, our static implementation is slightly slower than RTNeural. Overall, both of our convolutional layer implementations are, with a few exceptions, faster than WaveNetVA. Our dynamic version was slightly slower than WaveNetVA when the number of convolution channels exceeded 32.

In the same way, we also tested a convolutional layer with a kernel size of $M = 3$ and a dilation factor of $d = 2$. This is illustrated in Figure 6. We observed that our convolutional layers were the fastest in almost all cases. However, it can be seen that the advantage of a static implementation decreases rapidly with a higher number of convolution channels.

We did not perform further comparisons of the individual convolutional layers, but the following section outlines the signal processing speed-up using these layers in TCN models.

The source code for our implementation of convolutional layers was made publicly available at `https://github.com/stepanmk/FastTemporalConv`.

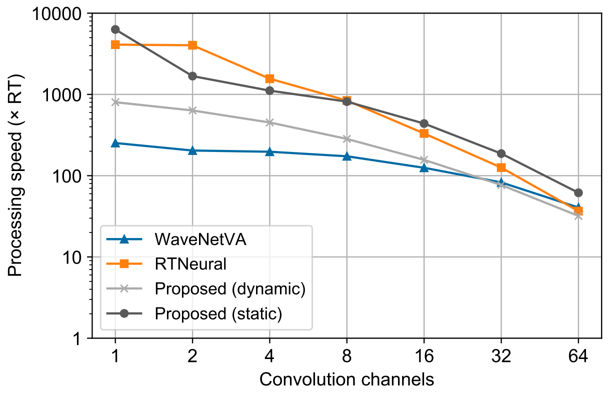

Figure 5: *The processing speed of a single convolutional layer with a kernel size of 1, and a dilation rate of 1.*

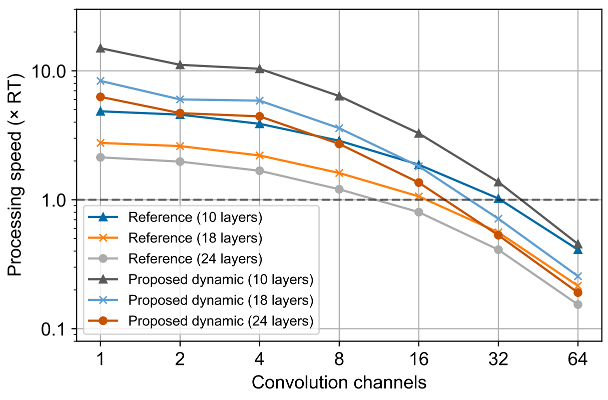

Figure 7: *Comparison of the reference TCN models and our dynamic implementation in terms of the processing speed. Models located below the dashed line were unable to operate in real time.*

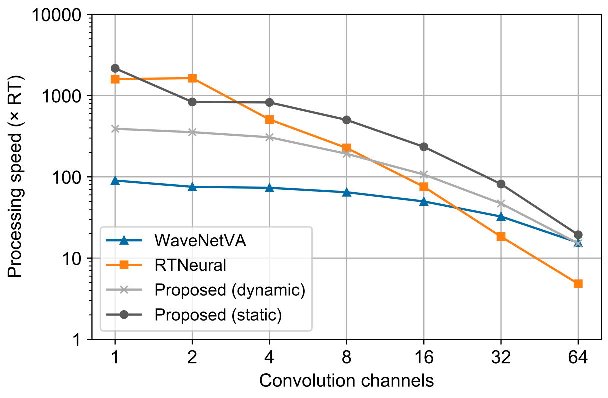

Figure 6: *The processing speed of a single convolutional layer with a kernel size of 3, and a dilation rate of 2.*

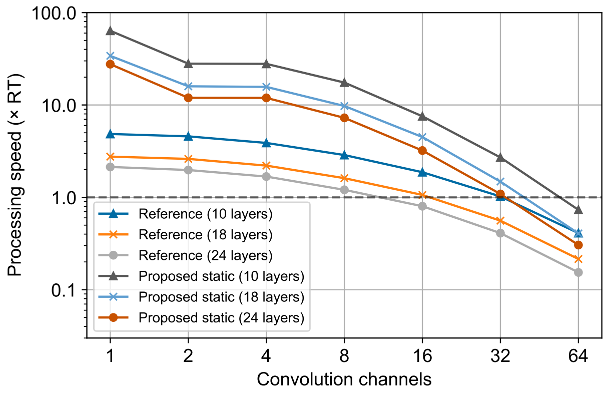

Figure 8: *Comparison of the reference TCN models and our static implementation in terms of the processing speed. Models located below the dashed line were unable to operate in real time.*

4.2. Neural Network Performance

When selecting the configuration of the TCN models, we followed paper [19] and considered three dilation patterns

$$d_k = \{1, 2, 4, \ldots, 512\},$$
$$d_k = \{1, 2, 4, \ldots, 256, 1, \ldots, 256\},$$
$$d_k = \{1, 2, 4, \ldots, 128, 1, \ldots, 128, 1, \ldots, 128\}.$$

The dilation patterns correspond to models with layer counts of 10, 18, and 24. All models had the convolution filter length set to $M = 3$.

During the validation of our dynamic implementation of TCN models, we found that it is possible to achieve roughly twice the signal processing speed compared to previous work. Our convolution algorithm yielded some improvement, but the signal processing speed was not dramatically increased. The results of this test are shown in Figure 7.

Next, we compared our static implementation with previous work. The results of this comparison are shown in Figure 8. The speed-up ranged from 2 to 12.8 times for models with the largest and lowest number of channels, respectively.

To get a better idea of the computational speed-up, we selected three configurations of TCN model hyperparameters as in [6]. The configurations are shown in Table 1. The models are deliberately ordered from least to most computationally expensive. Note that the TCN2 and TCN3 models are identical in structure except for a different number of convolution channels.

Table 1: *Hyperparameters of selected TCN models.*

Model	TCN1	TCN2	TCN3
Activation	Gated	Gated	Gated
Layers	10	18	18
Channels	16	8	16

Table 2 confirms that the most significant reduction in computation times was due to the static implementation of convolutional layers. Our static version of TCN1 was 4 times faster than the reference implementation in processing the signal. In the case of the TCN2 model, the processing speed was 6.2 times faster. Finally, for the TCN3, the processing speed was 4.2 times faster.

Table 2: *Compute times of 1 second of the audio signal using the reference and proposed models.*

Model	Ref.	Proposed (dynamic)	Proposed (static)
TCN1	0.53	0.30	**0.13**
TCN2	0.62	0.27	**0.10**
TCN3	0.93	0.54	**0.22**

Interestingly, the TCN2 model was marginally faster than the TCN1 model despite having a larger number of convolutional layers. This is because the TCN2 model has half the number of channels. However, in the case of the reference models, the lower channel count did not result in better performance of the TCN2.

We also compared the reference processing times from Table 2 with the times reported in [5, 6]. Our measured times for TCN1 and TCN2 were identical. The only difference we noted was with TCN3, as the authors report a slightly shorter processing time of 0.91 s.

4.3. Comparison with RNNs

Table 3 shows the processing times for RNNs with Gated Recurrent Units (GRU) and Long Short-Term Memory (LSTM) units adapted from [5]. Looking at the results from Table 2, we can observe that our static implementation of TCN models is similar in processing speed to RNNs described in previous work. In fact, the most complex TCN3 model is still approximately 1.8 times faster than the largest LSTM model with a hidden size of 96. In the case of the smallest TCN1 model, the processing speed is only 1.3 times slower than the fastest RNN model with GRUs and a hidden size of 32. We must also point out that the TCN2 model, which has eight more layers than the TCN1 model, achieved almost the same processing speed as the smallest RNN model.

Table 3: *Compute times of 1 second of the audio signal for the RNNs. The hidden size determines the computational complexity of the models.*

Model	Hidden size	Compute time (s)
GRU	32	0.097
LSTM	32	0.12
LSTM	64	0.24
LSTM	96	0.41

To further compare the RNN and TCN processing speed, we implemented our version of RNNs according to [5] to measure the compute times on our machine. We created two variants of these networks using static and dynamic variables, as was the case with TCNs. Figure 9 shows signal processing times of different RNN and TCN configurations. For the RNN models, the blue bars show the reference processing times reported in previous literature, and for the TCN models, these are our measured times of the WaveNetVA implementation. The orange and gray bars show the signal processing times using our RNN and TCN models with dynamic and static variables, respectively.

First, we note that our RNN models were significantly faster compared to the times reported in previous literature. We could not determine why this was the case because the source code of the original models was unavailable. Second, we found that using static variables does not contribute to noticeably faster processing times of RNNs because the mathematical operations within these models are done with matrices of too large dimensions. On the other hand, we must mention that a static implementation of TCN models makes sense, especially when the models are composed of layers with fewer convolution channels since the convolutions are performed by computing a Hadamard product of small matrices. Finally, due to our faster RNN implementation, our static TCN models are still slower than some of the smaller RNN networks. However, this is understandable since TCN models are composed of a large number convolutional layers.

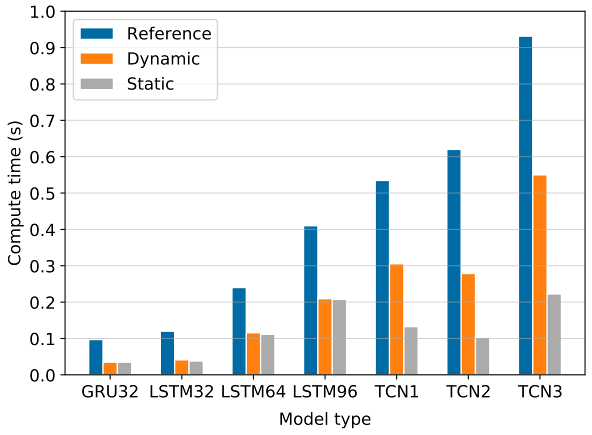

Figure 9: *Compute times of 1 second of the audio signal using different RNN and TCN models.*

5. CONCLUSIONS

In this paper, we presented a method for optimizing convolutional layers and neural network models for real-time audio signal processing. By implementing layers with fixed-sized variables, we achieved comparable processing times to state-of-the-art RNNs. The proposed algorithm can be deployed not only in the TCNs mentioned in this article but also in other variations of convolutional neural networks, such as those introduced in [20, 21]. It should also be possible to use simplified versions of these networks to model linear systems such as digital filters with arbitrary frequency responses.

The TCNs described in this and previous papers are quite complex and contain an excessive number of trainable variables. However, their main advantage is their use for black-box VA and audio effect modeling, where only basic assumptions are made about the internal structure of the devices in question. Nevertheless, there may be further room for optimization of these types of neural networks in terms of their structure. Modifications to these networks are beyond the scope of this article and are left as future work.

6. REFERENCES

[1] M. A. Martínez Ramírez and J. Reiss, "Modeling nonlinear audio effects with end-to-end deep neural networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP19)*, Brighton, UK, May 2019, pp. 171–175.

[2] S. Nercessian, A. Sarroff, and K. J. Werner, "Lightweight and interpretable neural modeling of an audio distortion effect using hyperconditioned differentiable biquads," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP21)*, Toronto, Canada, June 2021, pp. 890–894.

[3] T. Schmitz and J.-J. Embrechts, "Nonlinear real-time emulation of a tube amplifier with a long short time memory neural-network," in *Proc. Audio Eng. Soc. 144th Conv.*, Milan, Italy, May 2018.

[4] Z. Zhang, E. Olbrych, J. Bruchalski, T. J. McCormick, and D. L. Livingston, "A vacuum-tube guitar amplifier model using long/short-term memory networks," in *Proc. IEEE SoutheastCon*, Saint Petersburg, FL, April 2018.

[5] A. Wright, E.-P. Damskägg, and V. Välimäki, "Real-time black-box modelling with recurrent neural networks," in *Proc. Int. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, September 2019.

[6] A. Wright, E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-time guitar amplifier emulation with deep learning," *Appl. Sci.*, vol. 10, no. 3, Jan. 2020.

[7] A. Wright and V. Välimäki, "Neural modelling of periodically modulated time-varying effects," in *Proc. Int. Conf. Digital Audio Effects (DAFx-20)*, Vienna, Austria, September 2020.

[8] A. Wright and V. Välimäki, "Neural modeling of phaser and flanging effects," *J. Audio Eng. Soc.*, vol. 69, no. 7/8, pp. 517–529, Nov. 2021.

[9] M. Martínez Ramírez, E. Benetos, and J. Reiss, "A general purpose deep learning approach to model time-varying audio effects," in *Proc. Int. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, September 2019.

[10] M. A. Martínez Ramírez, O. Wang, P. Smaragdis, and N. J. Bryan, "Differentiable signal processing with black-box audio effects," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP21)*, Toronto, Canada, June 2021.

[11] M. A. Martínez Ramírez, E. Benetos, and J. D. Reiss, "Deep learning for black-box modeling of audio effects," *Appl. Sci.*, vol. 10, no. 2, Jan. 2020.

[12] J. Rämö and V. Välimäki, "Neural third-octave graphic equalizer," in *Proc. Int. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, September 2019.

[13] B. Kuznetsov, J. D. Parker, and F. Esqueda, "Differentiable IIR filters for machine learning applications," in *Proc. Int. Conf. Digital Audio Effects (DAFx-20in21)*, Vienna, Austria, September 2020.

[14] F. Eichas and U. Zölzer, "Black-box modeling of distortion circuits with block-oriented models," in *Proc. Int. Conf. Digital Audio Effects (DAFx-16)*, Brno, Czech Republic, September 2016.

[15] E.-P. Damskägg, L. Juvela, E. Thuillier, and V. Välimäki, "Deep learning for tube amplifier emulation," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process. (ICASSP19)*, Brighton, UK, May 2019, pp. 471–475.

[16] M. Holters and U. Zölzer, "A generalized method for the derivation of non-linear state-space models from circuit schematics," in *Proc. 23rd European Signal Processing Conf. (EUSIPCO)*, Nice, France, September 2015.

[17] F. Esqueda, B. Kuznetsov, and J. D. Parker, "Differentiable white-box virtual analog modeling," in *Proc. Int. Conf. Digital Audio Effects (DAFx20in21)*, Vienna, Austria, September 2021.

[18] A. van den Oord et al., "WaveNet: A generative model for raw audio," *arXiv preprint*, Sep. 2016, arXiv:1609.03499v2 [cs.SD].

[19] E.-P. Damskägg, L. Juvela, and V. Välimäki, "Real-time modeling of audio distortion circuits with deep learning," in *Proc. Int. Sound and Music Computing Conf. (SMC-19)*, Malaga, Spain, May 2019, pp. 332–339.

[20] C. J. Steinmetz and J. D. Reiss, "Efficient neural networks for real-time analog audio effect modeling," *arXiv preprint*, Jun. 2021, arXiv:2102.06200v1 [eess.AS].

[21] C. J. Steinmetz and J. D. Reiss, "Steerable discovery of neural audio effects," in *5th Workshop on Creativity and Design at NeurIPS*, 2021.

[22] S. Böck and M. E. P. Davies, "Deconstruct, analyse, reconstruct: How to improve tempo, beat, and downbeat estimation," in *Proc. of the 21st Int. Society for Music Information Retrieval Conf.*, Montréal, Canada, October 2020, pp. 574–582.

[23] K. Guirguis, C. Schorn, A. Guntoro, S. Abdulatif, and B. Yang, "SELD-TCN: Sound event localization & detection via temporal convolutional networks," in *Proc. 28th European Signal Processing Conf. (EUSIPCO)*, Montréal, Canada, October 2021, pp. 16–20.

[24] J. D. Parker, F. Esqueda, and A. Bergner, "Modelling of nonlinear state-space systems using a deep neural network," in *Proc. Int. Conf. Digital Audio Effects (DAFx-19)*, Birmingham, UK, September 2019.

[25] A. Peussa et al., "Exposure bias and state matching in recurrent neural network virtual analog models," in *Proc. Int. Conf. Digital Audio Effects (DAFx20in21)*, Vienna, Austria, September 2021.

[26] E.-P. Damskägg, L. Juvela, and V. Välimäki, "WaveNetVA," Available at https://github.com/damskaggep/WaveNetVA, accessed March 15, 2021.

[27] J. Chowdhury, "RTNeural: Real-time neural network inferencing," Available at https://github.com/jatinchowdhury18/RTNeural, accessed March 15, 2021.

[28] G. Gaël, J. Benoît, et al., "The Matrix class," Available at https://eigen.tuxfamily.org/dox/group__TutorialMatrixClass.html, accessed March 15, 2021.

[29] F. Eichas and U. Zölzer, "Gray-box modeling of guitar amplifiers," *J. Audio Eng. Soc.*, vol. 66, no. 12, pp. 1006–1015, Dec. 2018.