Audiomer: A Convolutional Transformer for Keyword Spotting

Surya Kant Sahu
The Learning Machines
surya.oju@pm.me

Sai Mitheran
National Institute of Technology, Tiruchirappalli
saimitheran06@gmail.com

Juhi Kamdar
George Mason University
jkamdar@gmu.edu

Meet Gandhi
George Mason University
mgandhi3@gmu.edu

Abstract

Transformers have seen an unprecedented rise in Natural Language Processing and Computer Vision tasks. However, in audio tasks, they are either infeasible to train due to extremely large sequence length of audio waveforms or reach competitive performance after feature extraction through Fourier-based methods, incurring a loss-floor. In this work, we introduce an architecture, Audiomer, where we combine 1D Residual Networks with Performer Attention to achieve state-of-the-art performance in Keyword Spotting with raw audio waveforms, outperforming all previous methods while also being computationally cheaper, much more parameter and data-efficient. Audiomer allows for deployment in compute-constrained devices and training on smaller datasets.

1 Introduction

The Transformer architecture [1], which mainly uses Self-Attention blocks, is shown to be very effective in Natural Language Processing and, recently, Computer Vision. However, the Self-Attention's memory and time complexity is quadratic w.r.t. the number of input tokens [2]. As a result, using a Vanilla transformer directly on a raw audio waveform is infeasible [1].

It is known that, given enough training data, neural networks can learn representations of low-level data such as pixels and learn to extract semantics that generalize better than human-designed features. This has led to the removal of human-designed features for visual tasks in state-of-the-art pipelines. However, such methods in the Audio domain i.e. sound event detection, Keyword Spotting [3] and Automatic Speech Recognition [4] still use Fourier-based methods to extract features due to the inability of Self-Attention to scale to large sequences. This has lead to information loss and hence a plateau in performance.

In this paper, we introduce *Audiomer*, a simple modification to a 1D ResNet [5], which leverages Performer Attention [6] between 1D convolution blocks. Consequently, our method is capable of learning from raw audio waveforms and achieve state-of-the-art results in a data, parameter and computationally efficient manner. On comparison with previous methods, our proposed uses only 0.18M parameters while outperforming transformer-based approaches which use over 5.36M parameters. We evaluate the performance of our architecture on the Keyword-Spotting task, using the *SpeechCommandsV2* (SC) [7] benchmark, which is released under the Creative Commons 4.0 BY license.

[1]A 5-second mono-audio clip sampled at 16kHz has 80,000 samples.

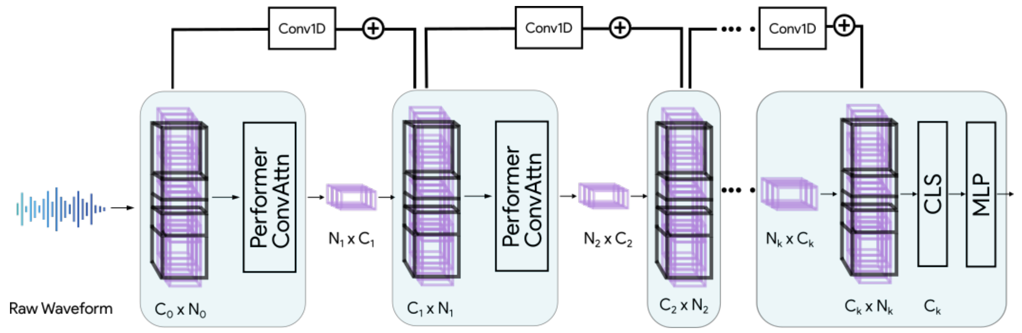

Figure 1: Overall Architecture of Audiomer. First, a CLS block is appended to the input waveform, it is then passed through successive *ConvAttention* blocks with residual connections. And finally, the first sample is passed to the Multi-Layer Perceptron which produces the logits.

2 Related Work

2.1 Convolutions in Transformers

A recent method, the Conformer [4] - which achieved state-of-the-art performance in Speech Recognition, asserted that the obtained performance boost was due to the inclusion of a convolution block, in addition to the Attention block. This alternative convolution path encourages locality bias, while the global attention of the transformer helps aggregate information past the local context of the convolution. Another recent work, CvT [8], which was introduced for Image Recognition, uses convolutional token embeddings as a first step to produce Keys, Queries, and Values rather than using an alternative path. We incorporate both of these ideas; however, we use Performer Attention [6], which is a variant of Self-Attention for which the memory and time complexity is linear rather than quadratic w.r.t sequence length. [8] further demonstrated through ablation experiments that positional encoding is redundant if the involved patches are overlapping, as in the case of our implementation, which motivates our architectural design.

In addition to this, [9] replaces the first few self-attention blocks in the Vision Transformer (ViT) [10] with convolutions to reach the pareto-frontier (Number of parameters v/s. Accuracy) on CIFAR10, claiming that their Compact Convolutional Transformer (CCT) is the most data-efficient vision-transformer as of yet, further aligning with our design choices.

2.2 Keyword Transformer

Keyword Transformer (KWT) [3] is a ViT-like transformer-based architecture for Keyword Spotting. KWT treats spectrogram-based features as 2D images and patches the features. This is followed by a stack of self-attention blocks and, finally, by pooling and an output projection. This simple architecture was found to be effective for Keyword Spotting. However, this architecture suffers from three significant problems: 1) Parameter and sample inefficiency 2) Inability to scale to longer audio due to quadratic complexity w.r.t sequence length; 3) Fixed maximum audio length, i.e., a trained model cannot be used with an audio clip of longer length than it was trained on due to the nature of positional encoding. In our work, we address these problems and propose an architecture that is more efficient in terms of parameters and compute; which can also scale to longer audio sequences.

3 Audiomer

In this section, we outline our proposed architecture - *Audiomer* and describe the design choices backed by an ablation study in Table 4. Fig. 1 provides a high-level representation of Audiomer. Audiomer is an amalgamation of ideas from state-of-the-art methods designed for Speech Recognition and Computer Vision tasks, crafted in a manner to improve model performance, while ensuring model efficiency.

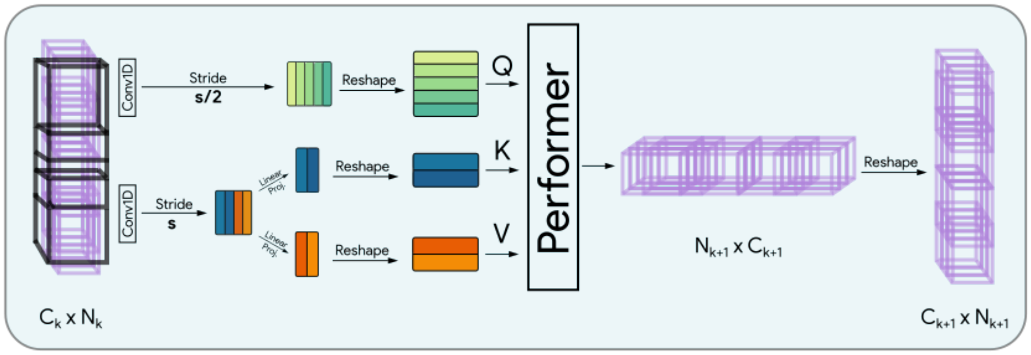

Figure 2: Illustration of the proposed *ConvAttention* module. A sequence with C_k channels and N_k samples is used to produce pre-Query and Context tensors, which are then used to produce the Query Q, Key K and Value V which are then used to compute the output tensor which is transposed.

3.1 ConvAttention Module

Recent work such as [8] have shown that overlapping token embeddings improve performance in the vision tasks; we introduce a novel method to produce key K, query Q, and value V using 1D convolutions over audio waveforms. *ConvAttention* has the following sub-modules:

- Squeezed 1D Convolutional Token embedding
- Performer Attention block
- Point-wise 1D Residual Convolution

Squeezed 1D Convolutional Token embedding Here, we define our Convolutional Token Embedding mechanism. Fig. 2 illustrates the *ConvAttention* module with the token embedding as the first step. We use two Squeeze-Excitation blocks with stride s and $s/2$ for producing the Context and pre-Query tensors respectively, followed by a Batch Normalisation operation in 1D, and finally a Point-wise 1D-Convolution with unit stride. Unequal strides ensure that the spatial resolution of keys and values is half of the spatial resolution of the queries. This saves some compute while improving performance marginally. [4] proposed to use a convolutional block over the incoming sequence as an alternative route to self-attention, this induced locality bias and was the primary reason of its state-of-the-art performance in ASR. [8] proposed a method for Image recognition, which uses convolution blocks to produce Key, Value and Query tensors prior to self-attention, and show that positional encoding can be discarded from the architecture because of overlapping tokens, which also allows the model to be used with inputs of different resolution than the inputs it was trained on. Hence, we choose the kernel size k such that $k > s$.

Performer Attention block We take the transpose of K, V, and Q from the previous step and feed it to the Performer block to generate the output sequence. The transpose operation is then applied on the output, to be fed to the next *ConvAttention* block. The Performer block has its own key, value, and query projection weights. We obtain K and V from the shared Context tensor, while Q is produced from the pre-Query tensor. Performer [6] has a Linear complexity (in both space and time) w.r.t the sequence length, while also having lesser number of parameters than Vanilla self-attention. This implies that our architecture can be scaled to long audio clips.

Performer uses an orthogonalized matrix for computing the attention matrix; however, this operation is slow to compute in most GPUs, adding a constant overhead. Hence, Performer is slower for smaller sequences but faster for longer sequences than Vanilla Self-Attention. This can be addressed by using alternative linear-attention mechanisms such as Linformer [11] or Fastformer [12] instead of Performer.

Pointwise 1D Residual Convolution We add a residual connection between the input to the convolution projection, and the attention-output, to improve the gradient flow. This residual connection

Table 1: Hyperparameters chosen for experiments. For data augmentations, we use TorchAudio-Augmentations [16] and apply the corresponding augmentations with given probabilities p for each training waveform.

Training	
Training Epochs	300
Batch size	128
Pooling	Novel CLS Pooling
Learning rate	$2e^{-3}$
β_1, β_2	0.9, 0.99
Scheduler	Cosine Annealing

Regularization	
Attention Dropout	0.2
MLP Dropout	0.2

Pre-processing	
Time window length	1 s
Sampling Rate	8192

Data augmentation	
Polarity Inversion	$p = 0.8$
White Noise	$p = 0.01$
Gain	$p = 0.3$
Reverb	$p = 0.6$

Table 2: Comparison of Keyword Transformer and Audiomer variants in terms of number of parameters, FLOPS, and size of the models.

Model	# Params	GFLOPS	Size (MB)
KWT-1	**0.61M**	**0.108**	**2.188**
KWT-2	2.39M	0.469	9.206
KWT-3	5.36M	1.053	20.523
Audiomer-S	**0.18M**	**0.061**	**0.987**
Audiomer-L	**0.8M**	**0.088**	**3.411**

adds minimal parameters but contributes significantly towards the performance, as evident from the ablation study in Table 4.

The time complexity of *ConvAttention* is $\mathcal{O}(knd)$, where n is the length of the input, k is the length of the filter, and d is the depth dimension. Here, n decreases at each stage, while d increases.

3.2 Classifier

Pooling We concatenate a learnable vector of size 128 to the beginning of the raw input waveform. This prepended block (referred to as CLS) is to maintain that the layers above aggregate classification-specific information at the beginning of the input sequence. After successive blocks of convolution and *ConvAttention*, we choose the first frame as input to the final Multi-Layer Perceptron (MLP) block, which produces the required logits. This method of pooling results in faster training and inference over mean-pooling, since a mean operation not required.

MLP We use a Multi-Layer Perceptron (MLP), a two-layer Linear (Fully-Connected) + ReLU [13] module with input and hidden dimensions equal to the hidden dimension of the last *ConvAttention* block; and output dimension equal to the number of classes.

4 Experiments

Datasets In order to evaluate the performance of our models, we used the *Google Speech Commands V2* (SC) dataset. It consists of $105,000$ recorded snippets containing audio of 35 unique keywords at 16kHz sampling frequency, each one second long. We train and test our model on the 12-label (SC-12), 20-label (SC-20), and 35-label (SC-35) classification tasks. To ensure fair comparison, we use the same train/validation/test split of $80 : 10 : 10$ as done in previous works [3, 14, 15].

Table 3: Comparison of our model on Google Speech Commands V2 (SC) datasets against the current state-of-the-art, which use complex techniques such as Knowledge Distillation and/or pre-training on external data. Our models outperform all baselines by a significant margin without pre-training or distillation. We provide mean test accuracy and 95% confidence computed after three trials. For baselines, we report the numbers from corresponding papers.

Dataset	Model	Extra Data	Knowledge Distillation	Accuracy
Speech Commands V2 12	KWT-3	✗	✓	98.56 ± 0.07
	KWT-2	✗	✓	98.43 ± 0.08
	KWT-1	✗	✓	98.08 ± 0.10
	Audiomer-S	✗	✗	**99.88** $\pm$ **0.08**
	Audiomer-L	✗	✗	**99.98** $\pm$ **0.02**
Speech Commands V2 20	Wav2KWS	✓	✗	97.80
	Audiomer-S	✗	✗	**99.85** $\pm$ **0.07**
	Audiomer-L	✗	✗	**99.90** $\pm$ **0.06**
Speech Commands V2 35	AST	✗	✗	98.11 ± 0.05
	AST	✓	✗	97.88 ± 0.03
	KWT-3	✗	✓	98.56 ± 0.09
	KWT-2	✗	✓	97.74 ± 0.03
	KWT-1	✗	✓	96.95 ± 0.14
	Audiomer-S	✗	✗	**99.44** $\pm$ **0.09**
	Audiomer-L	✗	✗	**99.74** $\pm$ **0.11**

Baselines We compare our method against Audio Spectrogram Transformer (AST) [17], Keyword Transformer (KWT) [3], and Wav2KWS [18]. Wav2KWS is a pre-trained Wav2Vec model fine-tuned on SC-12. AST is a pre-trained ViT that is fine-tuned on SC-35, and KWT is a ViT-like architecture trained using knowledge distillation with a Multi-Head Attention RNN (MHA-RNN) as the teacher model.

Model Variants We introduce two different variants of our proposed architecture, namely: *Audiomer-S*, and *Audiomer-L*. The variants differ based on the number of *ConvAttention* blocks and size of hidden dimension in each block. Audiomer-S has 9 blocks, with hidden dimensions: $[4, 8, 8, 16, 16, 32, 32, 64, 64]$; and Audiomer-L has 11 blocks, with hidden dimensions: $[4, 8, 16, 16, 32, 32, 64, 64, 96, 96, 192]$.

Various statistics related to model capacity are presented in Table 2. It can be observed that both our models are amongst the smallest and fastest, compared to other existing counterparts.

Implementation Details We implement our architectures and experiments[2] using PyTorch [19] and PyTorch Lightning [20]. We use a small kernel size of 5, with a stride of 4 for Context and 2 for pre-Query projections, respectively. In the Performer Attention block, we use 2 heads with hidden dimension 32 and Scale-norm after each transformation, and expansion factor of 2 for each feedforward block. We train the models for 300 epochs with stochastic averaging on the last 60 epochs, while validating thrice every epoch. We store the best checkpoints and evaluate them on the test set. All experiments where conducted on an NVIDIA Tesla P100 GPU with 16GB of VRAM for 2 days to generate results of a single trial. Other experimental details such as choice of optimizer and data augmentation configurations are provided in Table 1.

Results From Table 3, it can be concluded that our proposed architectures, Audiomer-S and Audiomer-L, outperform the previous best methods by a significant margin without pre-training or knowledge distillation, while using much lesser parameters. The superiority of our method is consistent across all three benchmarks. (SC-12, SC-20, and SC-35)

[2]Code will be released at `https://github.com/The-Learning-Machines/Audiomer-PyTorch`

It can be observed that, both, Audiomer-S and Audiomer-L achieve $\approx 1.5\%$ accuracy gain over KWT-3 on SC-12 and SC-35. Since the performance gap remains even on the larger SC-35 dataset, we infer that our method is more data-efficient than the Mel Frequency Cepstral Coefficients (MFCC) + Transformer approach of KWT.

Ablation Studies We provide an ablation study to demonstrate the importance of each module that constitutes our architecture. In each row in Table 4, we remove the following components from Audiomer, including the item removed in the previous rows, and then train and evaluate our model on the SC-12 dataset.

1. Squeeze Excitation in *ConvAttention*.
2. Squeezed Convolution Projection (Unequal strides) in *ConvAttention*.
3. Residual Connections through 1D Pointwise Convolution.

Table 4: This table illustrates the impact of successive removal of various key components on performance from Audiomer-S. For this experiment, we report accuracy after one trial.

Component	Accuracy
Audiomer-S	**99.91**
- Squeeze Excitation	**99.89**
- Unequal Strides	**99.86**
- Residual Conn.	**99.69**

It can be observed that each component plays an important role in increasing the performance of the proposed architecture, while also maintaining parameter and compute-efficiency.

5 Conclusion

In this paper, we introduce *Audiomer*, an efficient architecture for Keyword Spotting that is capable of processing raw audio waveform in the naturally available form. Our method enables users to avoid dependence on hand-crafted features, which incur a loss-floor. Audiomer uses 1D Convolution Projections along with Performer attention to learn rich representations of local and global audio context. Through experiments on Google Speech Commands v2 datasets, we show that Audiomer variants achieve state-of-the-art performance, outperforming by a considerable margin while having much fewer parameters and using lesser FLOPS. We intend this paper to inspire future work on building performant and efficient Transformer-based methods for Audio and Speech-related tasks in low data or on-the-edge use cases.

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[2] Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. Efficient transformers: A survey. *arXiv preprint arXiv:2009.06732*, 2020.

[3] Axel Berg, Mark O'Connor, and Miguel Tairum Cruz. Keyword transformer: A self-attention model for keyword spotting, 2021.

[4] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented transformer for speech recognition, 2020.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[6] Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamas Sarlos, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. Rethinking attention with performers, 2021.

[7] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition, 2018.

[8] Haiping Wu, Bin Xiao, Noel Codella, Mengchen Liu, Xiyang Dai, Lu Yuan, and Lei Zhang. Cvt: Introducing convolutions to vision transformers, 2021.

[9] Ali Hassani, Steven Walton, Nikhil Shah, Abulikemu Abuduweili, Jiachen Li, and Humphrey Shi. Escaping the big data paradigm with compact transformers. *arXiv preprint arXiv:2104.05704*, 2021.

[10] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[11] Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. Linformer: Self-attention with linear complexity, 2020.

[12] Chuhan Wu, Fangzhao Wu, Tao Qi, Yongfeng Huang, and Xing Xie. Fastformer: Additive attention can be all you need, 2021.

[13] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[14] Oleg Rybakov, Natasha Kononenko, Niranjan Subrahmanya, Mirko Visontai, and Stella Laurenzo. Streaming keyword spotting on mobile devices. *arXiv preprint arXiv:2005.06720*, 2020.

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

[16] Janne Spijkervet. Spijkervet/torchaudio-augmentations, 2021.

[17] Yuan Gong, Yu-An Chung, and James Glass. Ast: Audio spectrogram transformer, 2021.

[18] Deokjin Seo, Heung-Seon Oh, and Yuchul Jung. Wav2kws: Transfer learning from speech representations for keyword spotting. *IEEE Access*, pages 1–1, 2021.

[19] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *CoRR*, abs/1912.01703, 2019.

[20] et al. Falcon, WA. Pytorch lightning. *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, 3, 2019.