# LETR: A LIGHTWEIGHT AND EFFICIENT TRANSFORMER FOR KEYWORD SPOTTING

*Kevin Ding*[*], *Martin Zong*[*], *Jiakui Li*, *Baoxiang Li*[†]

SenseTime Research

## ABSTRACT

Transformer recently has achieved impressive success in a number of domains, including machine translation, image recognition, and speech recognition. Most of the previous work on Keyword Spotting (KWS) is built upon convolutional or recurrent neural networks. In this paper, we explore a family of Transformer architectures for keyword spotting, optimizing the trade-off between accuracy and efficiency in a high-speed regime. We also studied the effectiveness and summarized the principles of applying key components in vision Transformers to KWS, including patch embedding, position encoding, attention mechanism, and class token. On top of the findings, we propose the `LeTR`: a lightweight and highly efficient Transformer for KWS. We consider different efficiency measures on different edge devices so as to reflect a wide range of application scenarios best. Experimental results on two common benchmarks demonstrate that `LeTR` has achieved state-of-the-art results over competing methods with respect to the speed/accuracy trade-off.

***Index Terms*—** Transformer, keyword spotting, latency

## 1. INTRODUCTION

Transformer was originally proposed for machine translation [1]. Very recently [2], Vision Transformer (ViT) [3, 4, 5] has gained increasing interest due to its attention mechanism in enabling great flexibility in modeling long-range dependencies in vision tasks, and thereby has been considered to be a strong alternative to Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs). Transformers have also been explored for Keyword Spotting [6, 7, 8], reflecting its powerful representational and generalization ability.

However, the good performance of Transformers usually comes at a large model size and high computational cost. For example, a ViT-base/16 model [3] requires 86.6M parameters and 17.6G FLOPs. Such extremely high computational resources requirement is beyond the capabilities of many edge devices such as smartphones and IoTs. Therefore, it is of great importance to design a lightweight and efficient Transformer architecture tailored for real-world speech applications on the edge.

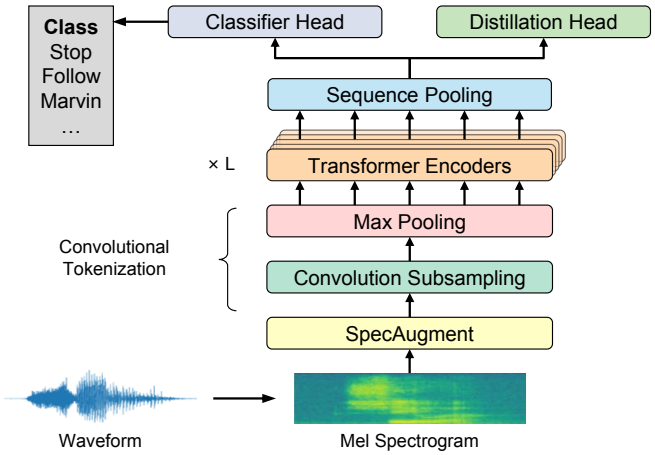


**Fig. 1**. Illustration of the proposed framework. Audio is transformed into a Mel-scale spectrogram with speech augmentation, then processed by a convolution subsampling layer and max-pooling layer. These form the input tokens for a multi-stage Transformer encoder. Then the sequence pooling layer pools the outputs of the Transformer backbone across the sequence. On top of the pooling results, we establish two MLP heads, one for classification and the other for distillation.

To this end, we explore the design space to offer even better trade-offs than CNN-based models in the regime of small Transformer architectures on keyword spotting. We are especially interested in optimizing the performance/accuracy trade-off, such as the latency and accuracy of processing a one-second speech input on an edge device. Drawing inspirations from the advantages of both Transformers and CNN architectures, we re-introduce convolutional components in place of Transformer components that learn convolutional-like features, and replace the uniform structure of a Transformer with a progressive shrinking pyramid [9] to reduce the computations of large feature maps. We thus develop a Transformer-based family of models for the KWS task with a lower memory footprint and better inference speed on highly parallel architectures such as GPUs, CPUs, and ARM hardware commonly found on mobile devices. Note that most hardware accelerators are optimized to perform large matrix multiplication, and in Transformer, attention and Multi-Layer

[*]Authors contributed equally

[†]Corresponding author: Baoxiang Li, libaoxiang@sensetime.com

Perceptron (MLP) blocks rely primarily on these matrix operations. In contrast, convolutions require complex data access patterns, leading to I/O bound operations. This implies that the Transformer architecture is more hardware-friendly than CNN, and these considerations incentivize us to explore the speed/accuracy trade-off.

To summarize, our contributions are as follows: ***i.*** We systematically analyze the effectiveness of applying the key components of Transformer architecture to the keyword spotting task. ***ii.*** We propose `LeTR`, as shown in Fig. 1, a lightweight and computationally efficient Transformer-based model for keyword spotting, integrating ingredients from both CNNs and Transformers. ***iii.*** We carefully compare our model on the Google Speech Commands dataset with a wide range of CNN, RNN, and Transformer baselines, and extensive experiments show its competitive performance over other competitors in terms of latency and accuracy.

## 2. RELATED WORK

### 2.1. Keyword Spotting

Keyword Spotting (KWS) refers to the task of detecting specific words of interest in audio signals, which has been an active research area in speech recognition for decades [8, 10, 11, 12]. To achieve this, audio is typically processed locally on edge devices such as smart speakers and mobile phones.

Early work used keyword/filter Hidden Markov Model (HMM) to establish KWS systems [10, 12]. Generally, an HMM model is trained for each keyword, and a filler model HMM is trained from the non-keyword segments of the speech signal (fillers). At runtime, these systems require Viterbi decoding, which can be computationally expensive depending on the HMM topology. Later, with the advent of deep learning, CNNs [11] and RNNs [7, 13] models for KWS have rapidly been increasing in popularity. These networks typically resort to a pre-processing pipeline that extracts the Mel-Frequency Cepstrum Coefficients (MFCC) [14], and directly predicts the keyword(s) or subword units of the keyword(s) followed by a posterior handling method producing a final confidence score.

### 2.2. Vision Transformer

Transformer was firstly proposed by [1] for machine translation tasks, and since then they have become dominant on a number of NLP tasks. Its core component is multi-head self-attention which is able to directly model the relationship between input tokens with arbitrary length in theory. Recently, Transformer was introduced for image classification, *e.g.*, ViT [3] and DeiT [5], and achieved compelling results compared with CNNs. They usually divide the images into patch sequences and feed them into a standard Transformer along with a class token. In addition, researches have shown that the integration of CNNs can improve the performance and efficiency of vision Transformers [15]. Transformers have also been explored for wake word detection [16], voice triggering [17], and keyword spotting. For example, KWT [8] devises a fully self-attentional architecture motivated by ViT [3] for keyword spotting. In contrast to these methods, our work focuses on tradeoffs between latency and efficiency in Transformer architecture design. Besides, we also put more emphasis on model computation and memory usage, so as to meet the needs of real-world keyword spotting scenarios.

## 3. METHOD

### 3.1. Model Architecture

**Convolutional Tokenization.** Given an input audio, we first extract the MFCC spectrogram, denoted as $\mathbf{x} \in \mathbb{R}^{\mathsf{T}\times\mathsf{F}}$, where $\mathsf{T}$ is the number of time window and $\mathsf{F}$ is the frequency. To introduce an inductive bias into the model, we replace the *patch embedding* in ViT [3] with a simple convolutional block. The convolutional block comprises a single convolution layer, a ReLU activation and a max-pooling operation. Formally, given $\mathbf{x} \in \mathbb{R}^{1\times\mathsf{T}\times\mathsf{F}}$, we have

$$\mathbf{x}_0 = \mathrm{MaxPool}(\mathrm{ReLU}(\mathrm{Conv2d}(\mathbf{x}))), \tag{1}$$

where $\mathbf{x}_0 \in \mathbb{R}^{\mathsf{d}\times\mathsf{h}\times\mathsf{w}}$ and the $\mathrm{Conv2d}$ operation has $c$ channels, same number as the embedding dimension of the Transformer backbone. The number of these convolutional blocks can exceed 1. When stacking multiple blocks, we set the number of channels for the earlier convolution operations to be 3, and for the final one to be $c$. This reduces the activation map fed into the Transformer without losing salient information.

**Multi-Resolution Pyramid.** In ResNet [18], convolutional architectures are built as pyramids, where the resolution of the activation maps decreases as their number of channels increases during processing. Inspired by this, we also integrate the ResNet stages within Transformer architectures. Specifically, inside each stage, `LeTR` consists of several alternated MLP and attention blocks. And between the stages of `LeTR`, a *Sub-Sampled Attention* (SSA) block reduces the size of the attention maps. More precisely, a average-pooling operations is applied before the $Q$ transformation in the self-attention mechanism (for building queries), which then propagates to the output of the soft activation. As illustrated in Fig. 2, given an input $\mathbf{x} \in \mathbb{R}^{\mathsf{c}\times\mathsf{h}\times\mathsf{w}}$, it returns an output tensor $\mathbf{x}' \in \mathbb{R}^{\mathsf{c}'\times\mathsf{P}\times\mathsf{P}}$ with $\mathsf{c}' > \mathsf{c}$, where $\mathsf{P}$ is the pooling size of the SSA.

**Relative Position Encoding.** The self-attention operation in Transformers is permutation-invariant, which cannot utilize the order of the tokens in an input sequence. To mitigate this issue, previous works [1, 3] add the absolute Positional Encoding (PE) to each token in the input sequence, which enables order-awareness. The PE can either be learnable parameters or fixed with sinusoidal functions, or can be imple-

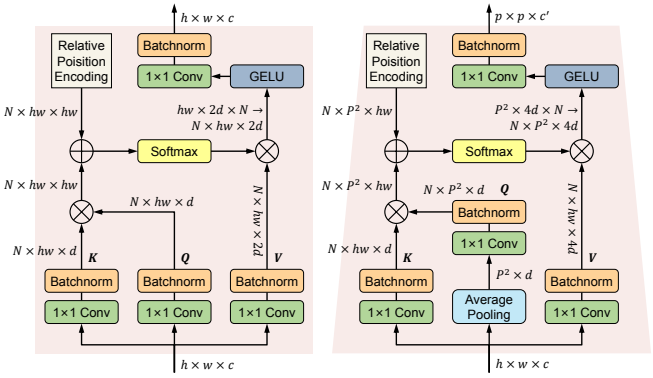


**Fig. 2**. LeTR attention blocks. *Left*: conventional attention block, *Right*: Sub-sampled attention block with reduction of *Query*, where $P$ is the pooling size and the input activation map is of size $c \times h \times w$. $N$ denotes the number of heads.

mented by 2-D convloutions with zero-paddings [19]. In our model, to make each attention block aware of position information and keep the desired translation-invariance in audios, we simply use the relative position encoding [20] on top of attention maps. Concretely, the scalar attention value between two pixels $(i, j) \in [\mathsf{h}] \times [\mathsf{w}]$ and $(i', j') \in [\mathsf{h}] \times [\mathsf{w}]$ for one head $h \in [N]$ is defined as

$$A^h_{(i,j)(i',j')} = Q_{(i,j),:} \cdot K_{(i',j'),:} + B^h_{|i-i'|,|j-j'|}, \quad (2)$$

where the first term is the vanilla attention, and the second one is the translation-invariant attention bias. Each head has $\mathsf{h} \times \mathsf{w}$ parameters corresponding to different pixel offsets.

**Reducing the MLP Blocks.** The MLP residual block in ViT transforms the embedding dimension by a factor 4, *e.g.*, $d \mapsto 4d \mapsto d$, using two dense layers and a GELU activation [21] as non-linearity. The MLP is more time-consuming than the attention block. Hence, we instantiate MLP with a 1×1 convolution, followed by batch normalization. To reduce the computational cost of that phase, we reduce the expansion factor of the convolution from 4 to 2, *i.e.*, $d \mapsto 2d \mapsto d$.

**Sequence Pooling.** Unlike ViT [3] and DeiT [5] that use a class token to yield the final class prediction, `LeTR` adopts the sequence pooling to gather the outputs of the Transformer backbone. Specifically, this operation constructs the mapping transformation $T : \mathbb{R}^{b \times n \times d} \mapsto \mathbb{R}^{b \times d}$. Given:

$$\mathbf{x}_S = \mathrm{f}(\mathbf{x}_0) \in \mathbb{R}^{b \times n \times d}$$

where $\mathbf{x}_S$ is the output of an $S$ stage Transformer encoder, $b$ is the mini-batch size, $n$ is the sequence length, and $d$ is the embedding dimension, we feed $\mathbf{x}_S$ to a linear layer $\mathrm{g}(\mathbf{x}_S) \in \mathbb{R}^{d \times 1}$ followed by softmax activation:

$$\mathbf{Att} = \mathrm{softmax}\left(\mathrm{g}(\mathbf{x}_S)^T\right) \in \mathbb{R}^{b \times 1 \times n}$$

Then, we obtain the output logits $z \in \mathbb{R}^{b \times d}$ by

$$z = \mathbf{Att} \times \mathbf{x}_S = \mathrm{softmax}\left(\mathrm{g}(\mathbf{x}_s)^T\right) \times \mathbf{x}_S \in \mathbb{R}^{b \times 1 \times d} \quad (3)$$

Then $z$ can be sent into a classifier head to infer the final keyword label. To alleviate the problem of data scarcity, we could simply add a distillation head on $z$, which resembles DeiT [5]. Specifically, let $Z_t$ be the logits of the teacher model, $Z_{sc}$ the logits of the student class head, and $Z_{sd}$ the logits of the student distillation head. We take the hard decision of the teacher as a true label, *i.e.*, $y_y = \arg\max_c Z_t(c)$. The overall objective associated with this hard-label distillation is as follows:

$$\mathcal{L}^{\mathrm{hardDistill}} = \frac{1}{2}\mathcal{L}_{\mathrm{CE}}(\psi(Z_{sc}), y) + \frac{1}{2}\mathcal{L}_{\mathrm{CE}}(\psi(Z_{sd}), y_{\mathrm{t}}), \quad (4)$$

where $y$ are the ground-truth labels, $\psi$ is the softmax function and $\mathcal{L}_{\mathrm{CE}}$ is the cross-entropy loss. During inference, the class and distillation head predictions are aggregated to infer the final keyword label.

## 4. EXPERIMENTS

We evaluate our model on the Google Speech Commands dataset V1 and V2 [22]. Both datasets comprise 1 second long audio snippets, sampled at 16 kHz, containing utterances of short keywords recorded in natural environments. Concretely, V1 contains 65,000 snippets of 30 different keywords, whereas V2 contains 105,000 snippets of 35 different keywords. The 30-label and 35-label classification tasks use all available words. We used the splits from [7, 22] for a fair comparison, where 80% instances for training, 10% for validation and 10% for testing.

**Training Details.** We take the same data pre-processing and augmentation strategy as in [7], including random time shifts, resampling, background noise, as well as augmenting the MFCC features using SpecAugment [23]. Following the setup in [8], we also choose the MhAtt-RNN [7] as the teacher and train it using the same augmentation of the input as the student. When involving the distillation, we mark our model as `LeTR`♣. As our model specializes in the high-throughout regime, we record the inference time on three different hardware platforms for a comprehensive evaluation, which are: *i.* A 32GB NVIDIA Tesla V100 GPU with 15.7 teraFLOPS top performance. *ii.* An Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz, which is typically deployed in a data center. *iii.* A Qualcomm Snapdragon 855 mobile platform, which contains a mobile CPU (based on ARM Cortex-A76) commonly seen in smartphones and other edge devices. On the GPU, we set the batch size to fit in memory and run timings on a batch. On the CPUs, we record inference time in a single thread, simulating a setting where several threads process separate streaming audios. More concretely, we use the PyTorch Mobile Benchmark Tools [1] to test model latency on the mobile device, where the latency is defined as the model's inference time for a given one-second audio input.

[1] https://pytorch.org/tutorials/recipes/mobile_perf.html

| Model | #stages | #blocks | #channels | #heads | #SSA: pooling size |
|---|---|---|---|---|---|
| LeTR-128t | 3 | [2,2,3] | [128,256,384] | [4,6,8] | [6,6] |
| LeTR-128s | 3 | [2,3,4] | [128,256,384] | [4,6,8] | [6,6] |
| LeTR-128 | 3 | [2,3,4] | [128,256,384] | [4,8,12] | [6,6] |
| LeTR-256 | 3 | [4,4,4] | [256,384,512] | [4,6,8] | [6,6] |

**Table 1**. The LeTR family of models. Each stage contains a varying number of pairs of attention and MLP blocks, where sub-sampled attention is used to connect the two stages.

| Architecture | Param. (M) | FLOPs (M) | V1 Acc. % | V1 GPU $\mu$s | V1 CPU ms | V1 ARM ms | V2 Acc. % | V2 GPU $\mu$s | V2 CPU ms | V2 ARM ms |
|---|---|---|---|---|---|---|---|---|---|---|
| TC-ResNet [24] | 0.365 | 41.8 | 96.60 | 4 | 0.6 | 5.4 | 95.43 | 4 | 0.6 | 5.2 |
| Att-RNN [6] | 0.534 | 80.9 | 95.63 | 91 | 12.2 | 22.1 | 96.90 | 98 | 13.0 | 23.5 |
| Matchbox [25] | 0.472 | 47.6 | 97.28 | 59 | 7.2 | 18.7 | 97.46 | 57 | 8.3 | 19.5 |
| EmbedHead [26] | 0.424 | 78.2 | 96.83 | 121 | 12.8 | 29.8 | 97.59 | 128 | 13.9 | 32.8 |
| MhAtt-RNN [7] | 0.743 | 101.9 | 97.26 | 143 | 20.4 | 36.8 | 97.27 | 156 | 21.7 | 39.2 |
| KWT-3 [8] | 5.361 | 246.6 | 97.24 | 222 | 28.0 | 84.3 | 97.51 | 230 | 29.2 | 88.4 |
| KWT-2 [8] | 2.394 | 155.6 | 97.36 | 134 | 16.3 | 45.4 | 97.53 | 142 | 16.6 | 46.8 |
| KWT-1 [8] | 0.607 | 50.6 | 97.05 | 59 | 7.4 | 20.1 | 96.85 | 61 | 7.5 | 20.7 |
| LeTR-128t | 0.402 | 37.0 | 97.33 | 39 | 4.5 | 15.2 | 97.62 | 41 | 4.7 | 15.9 |
| LeTR-128s | 0.452 | 40.2 | 97.57 | 50 | 6.6 | 17.8 | 97.66 | 53 | 6.9 | 18.5 |
| LeTR-128 | 0.603 | 48.2 | 97.61 | 53 | 7.0 | 18.7 | 97.82 | 53 | 7.1 | 18.7 |
| LeTR-256 | 1.106 | 79.6 | 97.85 | 84 | 12.0 | 33.6 | 98.04 | 88 | 12.4 | 35.1 |
| KWT-3♣ [8] | 5.371 | 250.5 | 97.49 | 228 | 28.7 | 86.5 | 97.69 | 237 | 30.0 | 89.1 |
| KWT-2♣ [8] | 2.399 | 158.5 | 97.27 | 135 | 16.4 | 46.3 | 97.74 | 141 | 17.4 | 48.4 |
| KWT-1♣ [8] | 0.610 | 51.4 | 97.26 | 65 | 8.1 | 22.2 | 96.95 | 68 | 8.3 | 22.3 |
| LeTR-128t♣ | 0.404 | 37.2 | 97.56 | 39 | 4.6 | 15.3 | 97.80 | 43 | 5.1 | 17.1 |
| LeTR-128s♣ | 0.454 | 40.4 | 97.70 | 52 | 6.9 | 18.4 | 97.89 | 54 | 7.1 | 19.1 |
| LeTR-128♣ | 0.607 | 48.6 | 97.79 | 58 | 7.7 | 20.7 | 98.06 | 59 | 7.8 | 20.9 |
| LeTR-256♣ | 1.112 | 80.7 | 97.94 | 88 | 12.6 | 35.3 | 98.15 | 91 | 13.0 | 36.8 |

**Table 2**. Performance comparison of LeTR and a wide range of strong competitors in terms of accuracy and latency, including CNN, RNN and Transformer-based methods.

**Model Variants.** The LeTR models can spawn a range of speed-accuracy tradeoffs by varying the size of the computation stages. We name them by the number of channels input to the first Transformer, e.g., LeTR-256 has 256 channels on the input of the Transformer stage. See more details in Table 1.

### 4.1. Comparison with State-of-the-Art

Table 2 records the mean accuracy and inference speed of different methods across three platforms on the Google Speech Commands V1 and V2 datasets. Specifically, the mean accuracy is measured at a 95% threshold as in [8], indicating that a decision will be made if the model confidence score exceeds it. From Table 2, we can see that our model outperforms the comparison methods by a large margin on both the V1 and V2 datasets. More precisely, LeTR has fewer parameters, lower computational complexity, and better accuracy than the Transformer baseline, *i.e.*, KWT. Besides, LeTR has significant advantages in inference speed on different platforms, even compared with the CNN-based (*e.g.*, Matchbox [25]) and RNN-based methods (*e.g.*, Att-RNN [6] and MhAtt-RNN [7]) with similar parameters. This can be speculated that the inherent attention mechanism hinges almost exclusively on matrix multiplications, which is hardware-efficient. Further, we observed that our low parameter models, *i.e.*, LeTR-128t, LeTR-128s, and LeTR-128 achieved comparable performance to MHAtt-RNN on the smaller challenging V1 dataset, but KWT-1 failed, revealing that introducing convolutions to Transformer is favorable for alleviating the data-hungry problem in KWT. Last, results also indicate LeTR can also benefit from the DeiT-like distillation, which makes it much more accurate when training on smaller datasets.

| Ablation Settings | #Params (M) | FLOPs (M) | Acc. (%) | ARM (ms) |
|---|---|---|---|---|
| Baseline model | 0.603 | 48.26 | 97.61 | 18.7 |
| w/o convolution | 0.619 | 51.86 | 95.45 $_{(-2.16)}$ | 20.5 |
| w/o pyramid struct. | 0.705 | 55.48 | 97.51 $_{(-0.10)}$ | 26.4 |
| w/o SSA | 0.647 | 51.53 | 95.82 $_{(-1.79)}$ | 22.5 |
| w/o RPE | 0.599 | 48.10 | 97.58 $_{(-0.03)}$ | 17.9 |
| w/o reducing MLP | 0.668 | 53.59 | 97.13 $_{(-0.48)}$ | 24.3 |
| w/o seqpooling | 0.683 | 54.38 | 96.42 $_{(-1.19)}$ | 25.2 |

**Table 3**. Ablation of key components w.r.t.the LeTR-128 baseline (baseline model). Each row represents a version of the baseline model minus some components.

### 4.2. Ablation Study

To find out what contributes to the performance of LeTR, we perform ablation studies on the V1 dataset by replacing the key components in LeTR with the counterparts in DeiT [5]. Table 3 shows that all changes unsurprisingly degrade the KWS accuracy. This evidences that the pyramid structure and the reduction of MLP blocks in LeTR are the important factors to reduce computational complexity. Replacing the convolutional tokenization with the patch-based tokenization causes subtle changes in the number of parameters, but dramatically increases the number of FLOPs, resulting in a strong performance drop. Allowing each attention head to learn a relative PE instead of adding PE to each token is also effective in improving the keyword spotting accuracy. Finally, we can see that sequence pooling enables the model to avoid the introduction of extra tokens, leading to fewer parameters and lower computational complexity.

## 5. CONCLUSION

In this work, we present LeTR, a lightweight and highly efficient architecture that incorporates ingredients from CNNs and Transformers for keyword spotting. We study the impact of each component and prove that pyramid structure, sub-sampled attention block, embedding dimension shrinkage and sequence pooling are effective in reducing computation and slimming model. Our model achieves higher accuracy with relatively fewer parameters and lower computational complexity than previous methods on benchmark datasets.

## 6. REFERENCES


[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017, pp. 5998–6008.

[2] K. Yuan, S. Guo, Z. Liu, A. Zhou, F. Yu, and W. Wu, "Incorporating convolution designs into visual transformers," *arXiv preprint arXiv:2103.11816*, 2021.

[3] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *ICLR*, 2020.

[4] D. Zhang, H. Zhang, J. Tang, M. Wang, X. Hua, and Q. Sun, "Feature pyramid transformer," in *European Conference on Computer Vision*, 2020, pp. 323–339.

[5] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image Transformers & distillation through attention," in *ICML*, 2021, pp. 10347–10357.

[6] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, "A neural attention model for speech command recognition," *arXiv preprint arXiv:1808.08929*, 2018.

[7] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," in *Interspeech*, 2020, pp. 2277–2281.

[8] A. Berg, M. O'Connor, and M. T. Cruz, "Keyword Transformer: A self-attention model for keyword spotting," *arXiv preprint arXiv:2104.00769*, 2021.

[9] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *CVPR*, 2017, pp. 2117–2125.

[10] J. G Wilpon, L. G Miller, and P Modi, "Improvements and applications for key word recognition using hidden Markov modeling techniques," in *ICASSP*, 1991, pp. 309–312.

[11] G. Chen, C. Parada, and G. Heigold, "Small-footprint keyword spotting using deep neural networks," in *ICASSP*, 2014, pp. 4087–4091.

[12] M. Sun, D. Snyder, Y. Gao, V. K Nagaraja, M. Rodehorst, S. Panchapagesan, N. Strom, S. Matsoukas, and S. Vitaladevuni, "Compressed time delay neural network for small-footprint keyword spotting," in *Interspeech*, 2017, pp. 3607–3611.

[13] R. Alvarez and H. Park, "End-to-end streaming keyword spotting," in *ICASSP*, 2019, pp. 6336–6340.

[14] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE TASLP*, vol. 28, pp. 357–366, 1980.

[15] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan, "Tokens-to-token vit: Training vision Transformers from scratch on imagenet," *arXiv preprint arXiv:2101.11986*, 2021.

[16] Y. Wang, H. Lv, D. Povey, L. Xie, and S. Khudanpur, "Wake word detection with streaming Transformers," in *ICASSP*, 2021, pp. 5864–5868.

[17] S. Adya, V. Garg, S. Sigtia, P. Simha, and C. Dhir, "Hybrid Transformer/CTC networks for hardware efficient voice triggering," in *Interspeech*, 2020, pp. 3351–3355.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[19] X. Chu, Z. Tian, B. Zhang, X. Wang, X. Wei, H. Xia, and C. Shen, "Conditional positional encodings for vision Transformers," *arXiv preprint arXiv:2102.10882*, 2021.

[20] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[21] D. Hendrycks and K. Gimpel, "Gaussian error linear units (gelus)," *arXiv preprint arXiv:1606.08415*, 2016.

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

[23] D. S. Park, W. Chan, Y. Zhang, C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," in *Interspeech*, 2019, pp. 2613–2617.

[24] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha, "Temporal convolution for real-time keyword spotting on mobile devices," in *Interspeech*, 2019, pp. 3372–3376.

[25] S. Majumdar and B. Ginsburg, "Matchboxnet: 1d time-channel separable convolutional neural network architecture for speech commands recognition," in *Interspeech*, 2020, pp. 3356–3360.

[26] J. Lin, K. Kilgour, D. Roblek, and M. Sharifi, "Training keyword spotters with limited and synthesized speech data," in *ICASSP*, 2020, pp. 7474–7478.