AudioRepInceptionNeXt: A lightweight single-stream architecture for efficient audio recognition

Kin Wai Lau[1,2], Yasar Abbas Ur Rehman[2], Lai-Man Po[1]
City University of Hong Kong[1]
TCL AI Lab[2]

***Abstract*—Recent research has successfully adapted vision-based convolutional neural network (CNN) architectures for audio recognition tasks using Mel-Spectrograms. However, these CNNs have high computational costs and memory requirements, limiting their deployment on low-end edge devices. Motivated by the success of efficient vision models like InceptionNeXt and ConvNeXt, we propose AudioRepInceptionNeXt, a single-stream architecture. Its basic building block breaks down the parallel multi-branch depth-wise convolutions with descending scales of $k \times k$ kernels into a cascade of two multi-branch depth-wise convolutions. The first multi-branch consists of parallel multi-scale $1 \times k$ depth-wise convolutional layers followed by a similar multi-branch employing parallel multi-scale $k \times 1$ depth-wise convolutional layers. This reduces computational and memory footprint while separating time and frequency processing of Mel-Spectrograms. The large kernels capture global frequencies and long activities, while small kernels get local frequencies and short activities. We also reparameterize the multi-branch design during inference to further boost speed without losing accuracy. Experiments show that AudioRepInceptionNeXt reduces parameters and computations by 50%+ and improves inference speed $1.28\times$ over state-of-the-art CNNs like the Slow-Fast while maintaining comparable accuracy. It also learns robustly across a variety of audio recognition tasks. Codes are available at https://github.com/StevenLauHKHK/AudioRepInceptionNeXt.**

***Index Terms*—CNN, Ausio recognition, Large kernel, Reparameterization**

I. Introduction

Learning deep feature representations for audio understanding has been extensively studied over the past decade using a variety of deep neural network architectures like Convolutional Neural Networks (CNN) [1]–[5], Long-Short Term Memory (LSTM) [6]–[9], and the recent Transformer networks [10]–[12]. These deep neural networks typically learn the mapping from an audio sample to its corresponding label, *intermediate* feature representations [13]–[15], or augmented audio sample [16], [17]. In practice, these deep neural networks for audio-understanding tasks have the flexibility to be trained by either using the raw audio samples [10], [11], [18] or a 2D time-frequency spectrogram [1], [3], [13]–[15], [19]–[23]. Recent advances in deep neural networks have had a revolutionary impact on numerous audio understanding domains, including but not limited to predictive tasks like sound event classification [24], the direction of voice prediction [25], speech command recognition [26], speaker identification [27], and generative tasks such as music generation [28]. Although the Transformers-based networks such as wave2vec [18] and ViT [10] show promising results in these audio understanding tasks; deploying them in their naïve form on the edge devices would require allocating massive amounts of compute resources for the architecture besides the audio data. For example, the base model of wave2vec 2.0 [18] requires over 89.78M (in millions) parameters compared to CNN models that only require 4.60M parameters [29]. This limits the applicability of Transformers in realizing numerous recent applications of general-purpose audio understanding that require on-device computation and training, such as federated learning [30]. Except for speech recognition, we found that the CNN-based deep neural networks are still prevalent for audio understanding tasks, such as audio event recognition and music classification, while maintaining similar performance compared to Transformers, and suitable for deployment and running on edge devices [31].

The Slow-Fast [1] is a recent framework focusing on the CNN architecture design for audio understanding, which proposed a two-stream pathway CNN, that has obtained better performance with lower parameter count than the Transformers on EPIC-SOUND datasets [31]. Following the success of separable kernels in the recent work on audio recognition [32], the Slow-Fast model proposed to use $1 \times k$ and $k \times 1$ kernels to capture the frequency and temporal feature independently considering non-homogeneous statistics of the audio-spectrogram. Later work [22] extends the study of the Slow-Fast model with self-supervised contrastive learning and discovered that such architectures provide better feature generalization on a diverse set of audio understanding tasks. Although performing remarkably well on a variety of audio understanding tasks, we found that these CNN models still incur high computational costs and memory footprints that potentially limit their applicability on edge devices for a variety of audio understanding applications. As an example, the Slow-Fast model incurs 26.68M parameters, which is $1.10\times$ higher than the conventional ResNet-50 model (with 24.13M parameters) (See Figure 1).

To enable general-purpose audio understanding on edge

K.W. Lau is with the Department of Electrical Engineering, City University of Hong Kong, Hong Kong, and also with TCL AI Lab. Y.A.U. Rehman is with TCL AI Lab. (e-mail: kinwailau6-c@my.cityu.edu.hk, yasar.abbas@my.cityu.edu.hk)

L.-M. Po is with the Department of Electrical Engineering, City University of Hong Kong, Hong Kong (email: eelmpo@cityu.edu.hk)

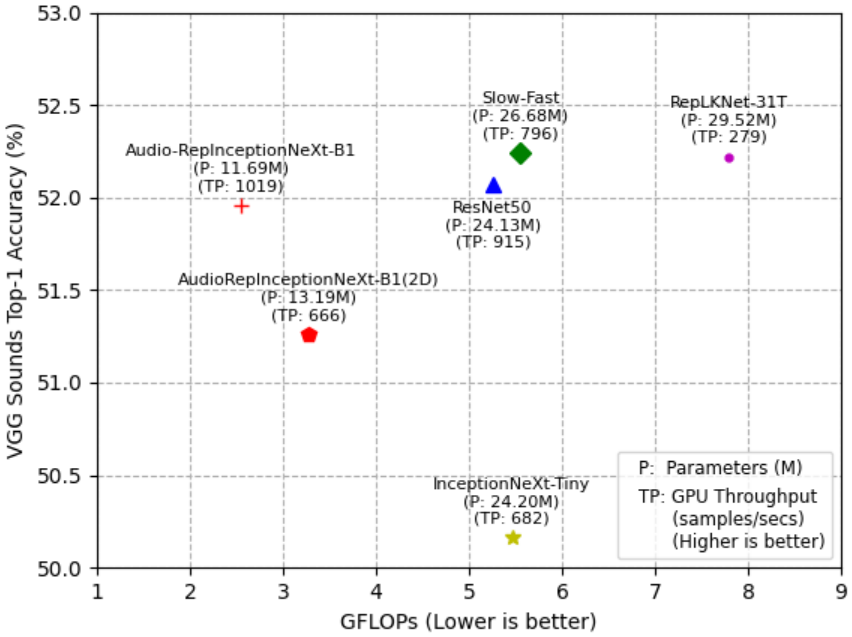


Fig. 1. Comparison of the Top-1 accuracy and GFLOPs on VGG Sounds. Different markers represent different baseline backbone architectures.

devices without incurring high computational and memory footprints, we focus here on a parallel rather unexplored area of redesigning and reparameterization of CNN architectures. Our motivation for redesigning and investigating the CNN-based architectures for general-purpose audio understanding is due to the following: (1) Although CNN architectures incur lower computational and memory costs, their performance heavily depends on the network architecture and design. In practice, direct deployment of the trained model on the low-end edge devices for inference might result in slow inference speed and increased memory footprints. (2) The Siamese networks, such as the Slow-Fast [1], incur a higher computational and memory footprint than single-stream networks like ResNet50, InceptionNeXt-Tiny and our proposed network (see. Figure 1). (3) The single-stream multi-branch CNN [31] networks with reduced parameters and theoretical GLOPS show low throughput (audio frames/seconds) due to the high memory access costs [33]–[35], and inefficient configuration of small operators in the multi-branch design [35].

To address these issues, we focus on the design of a very deep and parameter-efficient CNN architecture for general-purpose audio recognition tasks that can be easily deployed on edge devices. Unlike the Slow-Fast model proposed recently [1], we rethink the design of CNN for audio recognition and propose an efficient single-stream CNN architecture called AudioRepInceptionNeXt by employing parallel multi-scale separable convolutional kernels (see. Figure 2b).

The proposed model incurs lower computational and memory footprints while maintaining similar performance as the state-of-the-art CNN models. Additionally, the parallel multi-scale convolutional kernels in the proposed model can be rescaled to a single-scale convolutional branch during the inference time that not only further reduces the computational and memory footprints but also enhances the network throughput by a significant margin while maintaining the same performance as the original design. In this way, the model can capture the local and global temporal-frequency information via the multi-scale kernel designs during the training while eliminating the side-effect of multi-scale kernel design (i.e., slow inference speed and high memory access costs) during the inference time. Such design allows the simultaneous use of very large-scale kernels, e.g., 21×21, and small-scale kernels, e.g., 3×3 [33], [36], [37] in the parallel multi-scale branch of AudioRepInceptionNeXt during training. This design also allows our model to capture the global frequency semantic information and long-duration activities, and local details of frequency information and short-duration activities. We found that the proposed design takes few parameters (26.68M vs. 11.69M), lower computational complexity (5.55 GFLOPs vs. 2.55 GFLOPs), and higher inference speed (796 samples/sec vs. 1019 samples/sec) compared to the two-stream Slow-Fast model while achieving similar performance with a marginal difference of 0.28% in accuracy (see Figure 1).

Our contributions can be summarized as follow:

1) We address the computational inefficiency issues in the multi-stream Slow-Fast model. We show that the proposed single-stream multi-scale separable kernel architecture, AudioRepInceptionNeXt, effectively reduces the number of parameters and computational complexity incurred by multi-stream network architecture, without any performance degradation.

2) We employ reparameterization techniques [33], [34], [38] to further eliminate the side effect of multi-scale kernel design (i.e., slow inference speed and high memory access costs) by converting it into a single separable kernel during the inference, without any performance drop.
3) We validate the effectiveness of the proposed AudioRepInceptionNeXt on various audio classification tasks, including sound event classification, speech command classification, and music instrument classification. We demonstrate that AudioRepInceptionNeXt can achieve comparable or superior results as the state-of-the-art CNN-based models on the variety of downstream tasks [39] while saving half of the GFLOPs and memory footprint.

The rest of this paper is organized as follows. In Section II, we introduce literature on the state-of-the-art single-stream and multi-stream CNN-based network for audio recognition. In addition, we introduce the motivation for using model reparameterization in this paper. Section III presents our proposed AudioRepInceptionNeXt network, followed by the experiment results in Section IV. Section V provides ablation studies with the multi-branch design and the usage of large kernels. Finally, we conclude our work with a conclusion and future work in Section VI.

II. Related Work

A. Single-Stream Architecture to Multi-Stream Architecture

Single-stream convolution neural network (CNN) is widely used in audio recognition tasks including sound event classification, speech recognition, and music classification [2], [40]–[43]. There are two widely used approaches in single-stream CNN-based audio recognition systems. The first approach treats the raw audio waveform as input and utilizes a single-stream CNN to extract the feature for classification [5], [18], [43], [44]. Wav2vec [43] is a representation work in this regard that takes the 1D audio signal as input and trains a 1D CNN in an unsupervised contrastive manner to learn the *intermediate* audio representation for speech recognition. The second approach first preprocesses the 1D audio signal into a 2D time-frequency Mel-spectrogram followed by feeding it to the 2D CNN [3], [21], [45], [46].

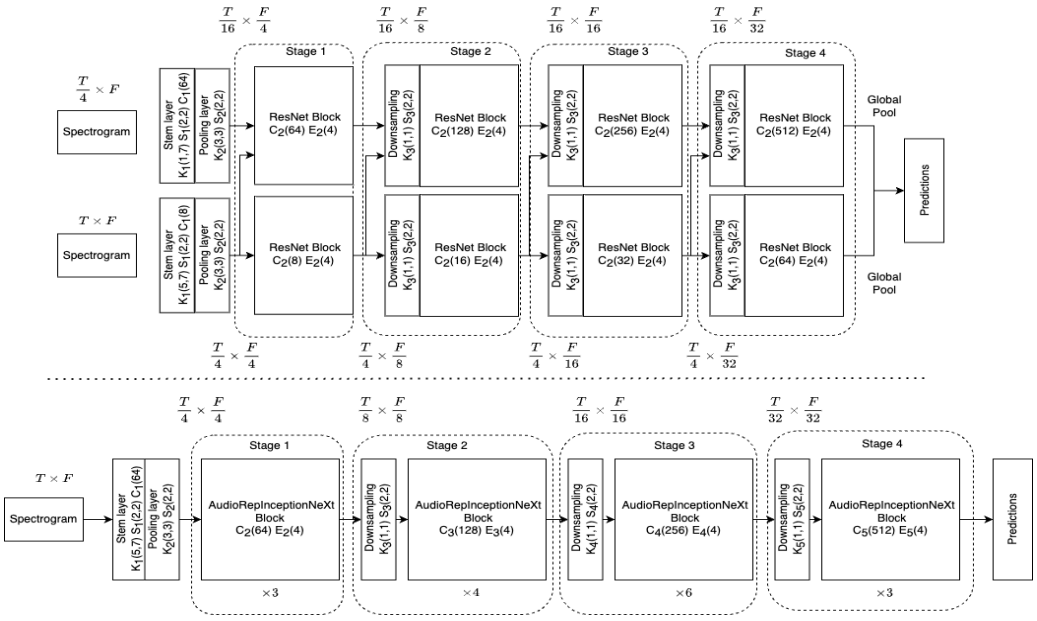
On the other hand, multi-stream architectures simultaneously work with either single or multimodal data such as raw audio and spectral features of audio [1], [23], [47]. Slow-Fast model [1] is a representative work that uses two-stream network architecture to capture both frequency and fine-grained temporal information from two different resolutions of the Log-Mel-Spectrogram independently. Their results demonstrated that the performance of the two-stream network outperforms the state-of-the-art single-stream network.

Although the above-mentioned single-stream and two-stream networks demonstrate promising results in audio recognition, they incur high computational and memory footprints during training and inference, which hinder their deployment on low-resource edge devices, such as smartphones. Therefore, it is imperative to explore a lightweight and parameter-efficient single-stream network that can achieve comparable results with the single-stream and two-stream networks while achieving low computational and memory footprints. In this work, we propose a parallel multi-scale convolutional kernel block to enrich the temporal and frequency feature representation during the training. This results in a single-stream CNN-based network architecture with lower parameters and computational footprints. The proposed design further enables the use of model reparameterization by combining multiple kernels into a single kernel during inference without any loss of performance.

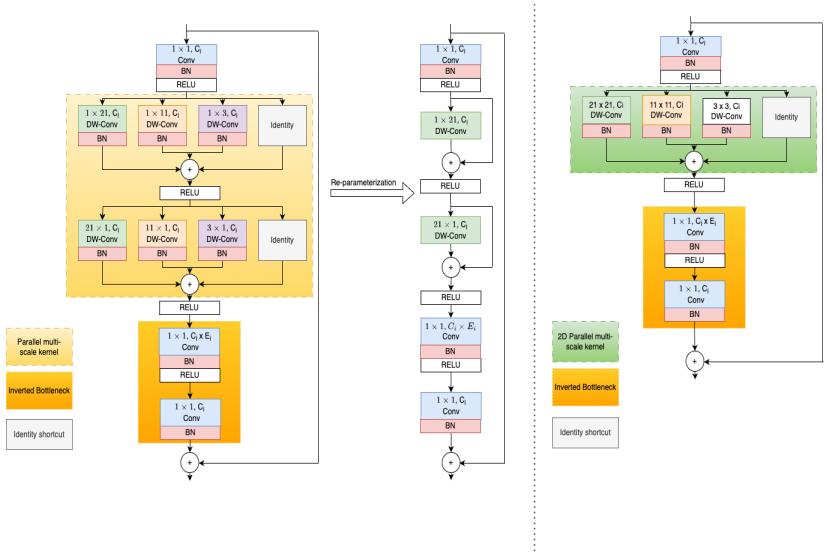
B. Model Re-parameterization

Model Reparameterization [33], [34], [38] approaches simplify complex multi-branch network architecture into a single-branch network structure during the model inference stage, without sacrificing performance. These approaches enable the use of the complex network architecture for learning efficient feature representations during the training stage and a simplified and parameters-efficient network during the inference stage. For instance, RepVGG [34] proposed extra 1×1 convolutional layers parallel to the 3×3 convolution layers in the original VGG network to learn richer feature representations during the training. After the training, the additional 1×1 layers are merged with the 3×3 layers via linear transformations of convolution, resulting in a VGG-like model with no extra parameters or computational cost during inference. RepVGG has been shown to outperform the original VGG model without adding any inference-time cost. Similarly, the Diverse Branch Block (DBB) method proposed in [38] enhances the representation capacity of a single convolution by combining multiple branches of varying complexities to enrich the feature representation. The DBB block includes a sequence of convolutions, multi-scale convolutions, and average pooling during training, which can be converted into a single convolutional layer using the linear transformation properties of convolution. Inspired by the success of RepVGG and DBB, RepLKNet [33] employed a similar technique by using a large kernel and a small kernel in parallel during training. After training, the small kernel is absorbed into the large kernel, enabling the large kernel to capture both global and local information resulting in improved the model's performance.

Our motivation for model reparameterization differs from these previous works in one important way. *We have discovered that the inference speed of the multi-branch architecture designs without reparameterization does not correlate closely with the number of model parameters and GLOPs* (see. Table II). For example, the multi-branch architecture designs before reparameterization, such as InceptionNeXt [48] and the proposed AudioRepInceptionNeXt, is slower compared to other models like the multi-stream Slow-Fast model, despite having lower parameters and GFLOPs. We attribute this discrepancy in speed to the increased memory access and synchronization time required by the multi-branch design [35]. Therefore, the reparameterization approach adopted in our work focuses on addressing the issue of optimizing the memory access and synchronization time of multi-branch designs. By optimizing



(a) Slow-Fast Model vs. AudioRepInceptionNeXt



(b) AudioRepInceptionNeXt Block (Left); AudioRepInceptionNeXt (2D) Block (Right)

Fig. 2. (a) Architecture of the Slow-Fast Model (Upper) and AudioRepInceptionNeXt (Bottom); (b) AudioRepInceptionNeXt Block during the training (Left) and Structural Re-parameterization of AudioRepInceptionNeXt Block after the training (Right); AudioRepInceptionNeXt (2D) (Right side of the dotted line) DW-Conv represents the depth-wise convolution and other notations can be found in Section III.

memory access and synchronization time in this way, the reduction in the number of model parameters and GLOPs becomes highly correlated with the increase in inference speed in multi-branch CNN design.

III. Methodology

In this section, we first describe the macro architecture design of the proposed AudioRepInceptionNeXt, followed by the micro block design in detail. We then provide a complexity analysis of the AudioRepInceptionNeXt Block.

A. Model Architecture

In this work, we use a typical hierarchical architecture design as depicted in Fig. 2 and the details are listed in Table I. The hyperparameters of the model are listed as follows:

- S_i: the stride used in the convolutional layer in the input stem and in the downsampling layers in stage i;
- K_i: the kernel size of the convolutional layer in the input stem and in the downsampling in stage i;
- C_i: the number of output channels of stage i;
- E_i: the channel expansion ratio of inverted bottleneck in stage i;
- L_i: the number of blocks in stage i;

1) Model Input: As depicted in Figure. 2a, AudioRepInceptionNeXt accepts the 2D audio mel-spectrogram as input with resolution $T \times F$. Unlike images, the width and height of the Mel-spectrogram represent distinct information, in which

TABLE I
DETAIL SETTINGS OF AUDIOREPINCEPTIONNEXT.

	Output Size	Layer name	AudioRepInceptionNeXt-B0	AudioRepInceptionNeXt-B1
Stage 1	$\frac{T}{4} \times \frac{F}{4}$	stem	$S_1 = (2,2); K_1 = (5,7)$	
		Max pooling layer	$S_2 = (2,2); K_2 = (3,3)$	
		Embed. Dim	$C_1 = 32$	$C_1 = 64$
		Convolution Encoder	$C_2 = 32$ $L_2 = 3$ $E_2 = 4$	$C_2 = 64$ $L_2 = 3$ $E_2 = 4$
Stage 2	$\frac{T}{8} \times \frac{F}{8}$	Downsampling	$S_3 = (2,2); K_3 = (1,1)$	
		Convolution Encoder	$C_3 = 64$ $L_3 = 4$ $E_3 = 4$	$C_3 = 128$ $L_3 = 4$ $E_3 = 4$
Stage 3	$\frac{T}{16} \times \frac{F}{16}$	Downsampling	$S_4 = 2; K_4 = (1,1)$	
		Convolution Encoder	$C_4 = 128$ $L_4 = 6$ $E_4 = 4$	$C_4 = 256$ $L_4 = 6$ $E_4 = 4$
Stage 4	$\frac{T}{32} \times \frac{F}{32}$	Downsampling	$S_1 = (2,2); K_1 = (1,1)$	
		Convolution Encoder	$C_5 = 256$ $L_5 = 3$ $E_5 = 4$	$C_5 = 512$ $L_5 = 3$ $E_5 = 4$

T and F axes correspond to the time and frequency bin, respectively. The time axis is generally longer than the frequency axis.

2) Marco Design: Adhering to the design of ResNet50 [49], our model comprises an input stem layer and four subsequent AudioRepInceptionNeXt stages. The first stem layer consists of a 5×7 convolutional layer with a stride of 2 on both the time and frequency axis and output feature maps with 64 channels. It is followed by a 3×3 Max-pooling layer with stride 2. The spatial resolution of the output feature maps after these two layers is 4 times lower compared to the spatial resolution of input to the network. Except for stage 1, each stage starts with the downsampling convolutional layer having 1×1 kernel and a stride of 2. The downsampling layer is subsequently followed by an AudioRepInceptionNeXt block that contains a 1×1 convolution layer and a parallel separable kernel with kernel sizes of 21, 11, and 3. Similar to ConvNeXt [37], we adopted an inverted bottleneck design in each of the AudioRepInceptionNeXt blocks, where the width of the 1×1 MLP layer (expansion layer) is four times wider (along the channel dimensions) than the width of the input (along the channel dimension), as shown in Fig.2b (left).

B. AudioRepInceptionNeXt Block

In this section, we describe the main components of the AudioRepInceptionNeXt block.

1) Parallel multi-scale kernel: As mentioned in Section I, the key property of the Slow-Fast model is utilizing two streams of the CNN network in parallel to capture the temporal and frequency information at different scales. In contrast, we adopted the multi-branch design, motivated by the visual CNN-based InceptionNet [50], which allows us to capture the temporal and frequency information at multiple scales with a single stream network. However, unlike the InceptionNet, we aim to use large-size kernels in the multi-branch layers, e.g., 21×21 and 11×11, in conjunction with small-size kernels, e.g., 3×3 and 1×1. The large kernel (i.e., 21×21 and 11×11) captures the global-frequency semantic information and long-duration activities, while the small kernel (i.e., 3×3) captures the local-frequency information and short-duration activities. Finally, all the feature maps are added and passed to the 1×1 layers for channel-wise information exchange. However, naïvely using the large kernels may incur high computational costs. To tackle this issue, we use depthwise separable convolutional kernel design as explained in the following subsection.

2) Depthwise Separable Kernel: As the parallel large kernel convolutional layers incur high computational costs in terms of the number of network parameters, we adopt the separable kernel design with depth-wise convolution (DW-Conv) following [1], [32]. It should be noted that our design of depthwise separable kernels refers to the *decomposition* of the 2D depthwise kernel and not to the conventional 2D depthwise separable kernels [51] (that employ 2D depthwise convolution following the 1×1 convolution). We decompose the 2D depthwise convolutional kernels, $k \times k$, into $1 \times k$ and $k \times 1$ as shown in Fig.2b (left). However, instead of using a cascaded $1 \times k$ and $k \times 1$ architecture design, we first aggregate the multi-scale temporal features obtained by applying $1 \times k$, followed by aggregating the multi-scale frequency features obtained by applying the $k \times 1$ kernel. This design helps in obtaining two separable kernels $1 \times k$, and $k \times 1$ after applying the reparameterization technique resulting in faster inference speed. In addition to reducing the memory footprint and computational time, previous studies [1], [32] have demonstrated that the use of such types of separable kernels allows the model to extract temporal and frequency information independently, leading to improvements in audio classification tasks. The reason is that the statistics of the spectrogram are not homogeneous, unlike natural images. We conducted a similar experiment within our network to verify the advantages of utilizing separable kernels, as discussed in Section IV-F.

3) Inverted Bottleneck: In the conventional design of the inverted bottleneck [52], [53], the number of channels in the hidden layer was four times the input channels. However, with large kernels e.g., 21, and 11, this design incurs increased computational cost. Unlike these methods, we place the parallel multi-scale depthwise separable kernel at the top before

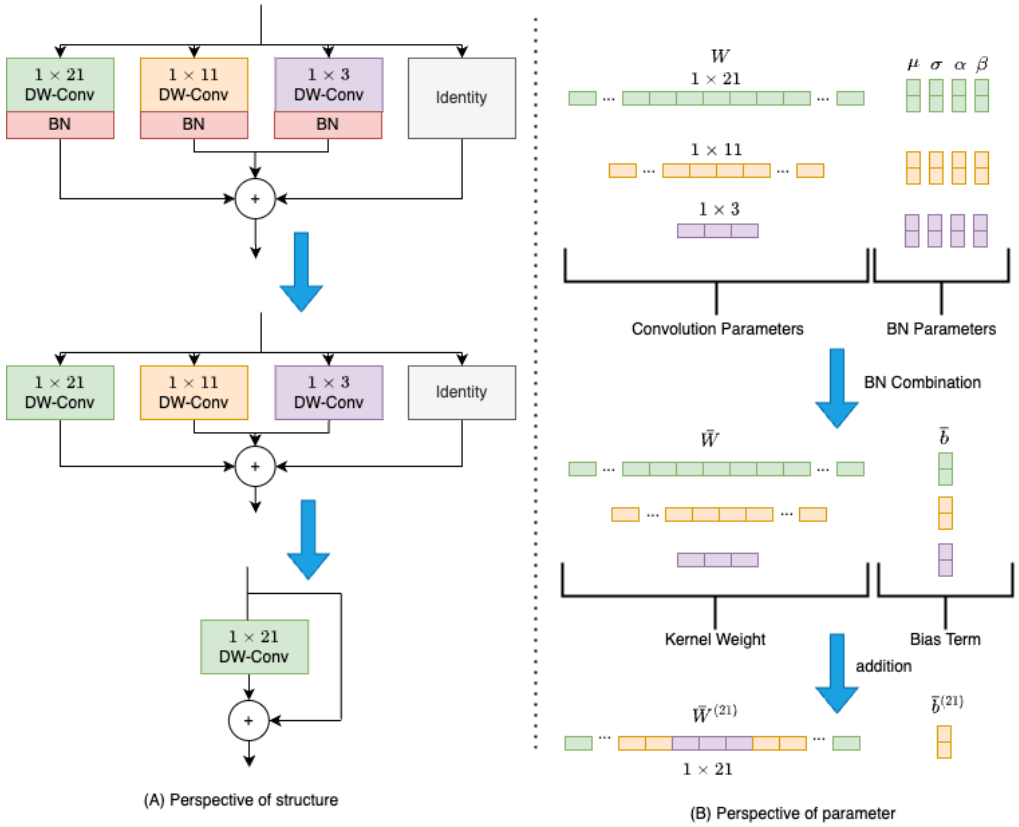


Fig. 3. Re-parameterization of the horizontal multi-scale kernel in the AudioRepInceptionNeXt Block. Here we assume all the layers have the same number of input channels, output channels, and stride size.

applying the 1×1 expansion layer with expansion ratio E_i for channel-wise information exchange following the ConvNeXt design.

4) Identity shortcut: Shortcut connections make the model an implicit ensemble of numerous shallower models [33], [54], such that the model can benefit from the different receptive fields. We demonstrate that shortcuts can improve the performance of AudioRepInceptionNeXt by 0.67% in VGG-Sound audio event classification [39] as shown in section V.

C. Reparameterization for Inference time model

We employ the AudioRepInceptionNeXt depicted in Figure 3(a) during the training stage to learn feature representations from audio signals. During the inference stage, we first apply the reparameterization technique to convert the multi-branch AudioRepInceptionNeXt blocks into a single-branch reparametrized block as shown in Figure 3(b).

In this subsection, we describe how to convert the trained multi-branch kernels with varying scales into a single kernel (i.e., $1 \times k$ and $k \times 1$) for inference as shown in figure 3(b). Here we use the horizontal $1 \times k$ kernels as an example. A similar operation can be applied to the vertical $k \times 1$ kernel. One can see from Figure 3(b) we use three horizontal kernels of size 1×21, 1×11, and 1×3. Specifically, we use $W^{(21)} \in \mathbb{R}^{C_{in} \times C_{out} \times 1 \times 21}$ to denote the kernel of a 1×21 convolution layer with C_{in} input channels and C_{out} output channels, $W^{(11)} \in \mathbb{R}^{C_{in} \times C_{out} \times 1 \times 11}$ and $W^{(3)} \in \mathbb{R}^{C_{in} \times C_{out} \times 1 \times 3}$ for kernel 1×11 and 1×3, respectively. We use $\mu^{(i)}$, $\sigma^{(i)}$, $\alpha^{(i)}$, and $\beta^{(i)}$, where $i \in 3, 11, 21$, as the mean, standard deviation, learned scaling factor and bias of the batch normalization (BN) layer following the 1×21, 1×11 and 1×3 convolution layers. Let $F^{(1)} \in \mathbb{R}^{N \times C_{in} \times H_1 \times W_1}$ and $F^{(2)} \in \mathbb{R}^{N \times C_{out} \times H_2 \times W_2}$ be the input and output features of the multi-scale horizontal convolution layers, respectively. N, H and W represent the batch size, height and width of the feature map, respectively. We assume $C_{in} = C_{out}$, $H_1 = H_2$, and $W_1 = W_2$ for simplifying the calculation. Before the re-parameterization, the output of the multi-branch horizontal convolution layers can be obtained by using the following equation.

$$\begin{aligned} F^{(2)} = & BN\{F^{(1)} * W^{(21)},\ \mu^{(21)},\ \sigma^{(21)},\ \alpha^{(21)},\ \beta^{(21)}\} \\ & + BN\{F^{(1)} * W^{(11)},\ \mu^{(11)},\ \sigma^{(11)},\ \alpha^{(11)},\ \beta^{(11)}\} \\ & + BN\{F^{(1)} * W^{(3)},\ \mu^{(3)},\ \sigma^{(3)},\ \alpha^{(3)},\ \beta^{(3)}\}, \end{aligned} \tag{1}$$

where

$$BN(F, \mu, \sigma, \alpha, \beta)_{:,j,:,:} = (F_{:,j,:,:} - \mu_j)\frac{\alpha_j}{\sigma_j} + \beta_j. \tag{2}$$

Note that the identity branch is ignored in the equation 1 for simplifying the calculation. In equation 1, the $*$ represents the convolution operation. In equation 2, $BN(.)$ and j represent the batch normalization function and output channel index, respectively. $F_{:,j,:,:}$ represents the j^{th} feature map output by the layer preceding the batch normalization layer. Note that the

sub-indices of the $BN(.)$ function and F follow the following order: batch size, output channel, height of the feature map, and width of the feature map. We first convert every BN layer and its corresponding horizontal convolution layer into a single convolution layer with a bias term. Let $\bar{W}^{(i)}$ and $\bar{b}^{(i)}$ be the kernel and the bias term after the combination, respectively. The kernel weight and bias can be obtained via the following equation.

$$\bar{W}^{(i)}_{j,:,:,:} = \frac{\alpha_j}{\sigma_j} W_{j,:,:,:}, \tag{3}$$

$$\bar{b}^{(i)}_{j} = -\frac{\mu_j \alpha_j}{\sigma_j} + \beta_j. \tag{4}$$

Note that the sub-indices of the kernel weight $\bar{W}^{(i)}_{j,:,:,:}$ follow the following order: output channel, input channel, height of the kernel, and width of the kernel. After the combination, the output of each branch can be obtained by the following equation.

$$\begin{aligned} &BN(F^{(1)} * W^{(i)}, \mu^{(i)}, \sigma^{(i)}, \alpha^{(i)}, \beta^{(i)})_{:,j,:,:} \\ &= (F^{(1)} * \bar{W}^{(i)})_{:,j,:,:} + \bar{b}^{(i)}_{j}. \end{aligned} \tag{5}$$

After the combination of the BN layer and its corresponding convolution layer, we can obtain three convolutional layers with horizontal kernel and three bias terms. Then we obtain the final 1×21 kernel by adding the 1×11 and 1×3 onto the central point of the 1×21 kernel and the final bias term by adding three bias terms together. The final kernel weight and bias can be obtained by the following equation.

$$\bar{W}^{(21)}_{j,:,:,:} = \sum_{n=1}^{i} \frac{\alpha^n_j}{\sigma^n_j} W^n_{j,:,:,:}, \tag{6}$$

$$\bar{b}^{(21)}_{j} = \sum_{n=1}^{i} -\frac{\mu^n_j \alpha^n_j}{\sigma^n_j} + \beta^n_j. \tag{7}$$

Before adding both the small kernels to the large kernel, we apply zero padding to both small kernels such that they have the same kernel size as 1×21. Note that such transformation requires the 1×11 and 1×3 layer to have the same stride.

D. Comparison to Multi-stream Slow-Fast model

Compared to the current state-of-the-art CNN-based Slow-Fast model [1], our new proposed network uses a single-stream architecture instead of two-stream architecture. Our 21×1 and 1×21 large separable kernel can focus on the global-frequency semantic information and long-duration activities inherent in the similar functionality of the Slow model. Meanwhile, the 3×1 and 1×3 separable kernels act as a Fast model that captures the local-frequency semantic information and short-duration activities. The major benefit of using the single-stream network is that the computational complexity and memory footprint can be reduced by 54% and 56%, respectively, compared to the two-stream slow-fast model. In contrast, our network can achieve comparable performance as the slow-fast model. The following section will provide the complexity analysis of the new proposed architecture.

IV. Experiment

A. Pretraining Dataset

VGG-Sound. VGG-Sound [39] is a large-scale audio dataset extracted from YouTube videos. It contains more than 200K audio clips each with 10 seconds duration sampled at 16KHz. There are a total of 309 classes which include the sound emitted from objects, human actions, and interactions.

B. Downstream Task Datasets

EPIC-KITECHENS-100. EPIC-KITECHENS-100 [55] is a large-scale egocentric audio-visual dataset, which captures daily activities in the kitchen. The videos are being recorded in 45 different kitchens and contain 100 hours of data. It includes 90K trimmed action clips and they capture the hand-object activities. The ground-truth labels are formed by a verb and a noun (e.g., move tap, open kettle, and open bin). There are 97 verb classes and 300 noun classes in total. Most of the actions are short-duration (average action length is 2.6 seconds). The audio is sampled at 24kHz.

EPIC-Sound. EPIC-Sound [56] is a large-scale audio event classification dataset that is re-annotated from the EPIC-KITCHENS-100 dataset. Unlike EPIC-KITCHENS-100, the actions in EPIC-Sound can be discriminated purely from the audio, for example, cutting food instead of cutting tomatoes. This dataset includes 75.9k segments of audible events and actions with 44 classes.

Speech Commands V2 (KS2). Speech Commands V2 [57] is an automatic speech commands recognition dataset that contains more than 100K audio clips with 1 second for each of them. It also contains 35 common speech commands for the recognition task.

UrbanSound8K (Urban8K). UrbanSound8K [58] is a urban sound event classification dataset. It contains 8K labeled sound excerpts with less than 4 seconds for each clip and 10 urban sound classes.

NSynth. Nsynth [4] is an audio dataset containing 305,979 musical notes and each of them with a unique pitch, timbre, and envelope. The sounds were collected from 1006 instruments from commercial sample libraries and annotated based on their source, instrument family, and sonic qualities. There are 11 instrument families in total.

C. Training and Validation Details

We follow the same training strategy as the baseline Slow-Fast model [1], i.e., we pretrain our models on the VGG-SOUND dataset [39] followed by fine-tuning it on the downstream tasks datasets. The input audio signal is first converted into a Log-Mel spectrogram with 128 Mel bands before feeding it to the network. During both the pretraining and fine-tuning stages, we applied the augmentation methods proposed in SpecAugment [17] following the common practice as in [1], [10]–[12]. These augmentations include frequency masking, time masking, and time warping. All the models are trained with a batch size of 32 on 4x NVIDIA RTX3090 GPUs during the pretraining and fine-tuning stages.

TABLE II
COMPARISON WITH CNN-BASED SOTA METHODS ON VGG-SOUND EVENT CLASSIFICATION PRETRAINING DATASET. REPARAM STANDS FOR RE-PARAMETERIZATION. PARAM STANDS FOR PARAMETER SIZE. GFLOPS STANDS FOR FLOATING POINT OPERATIONS. TP STANDS FOR THROUGHPUT. TOP-1 STANDS FOR TOP-1 ACCURACY, TOP-5 STANDS FOR TOP-5 ACCURACY, MAP STANDS FOR MEAN AVERAGE PRECISION AND AUC STANDS FOR AREA UNDER CURVE. NOTE THAT THE MODELS ARE SEPARATED BY THE MODEL SIZE. THE UPPER THREE ROWS REPRESENT THE SMALL MODEL SIZE, WHILE THE REMAINING ROWS REPRESENT THE LARGE MODEL SIZE.

Model	Reparam (M)	Param (M)	GFLOPs	TP (GPU)	TP (CPU)	Top-1	Top-5	mAP	AUC	d-prime
BCResNets-8 [59]	N/A	0.39	1.45	492	5.62	46.42	73.70	35.6	95.8	2.45
BN-InceptionNetV1 [60]	N/A	6.35	1.86	2031	12.10	**50.85**	**77.20**	**53.2**	97.3	2.73
AudioRepInceptionNeXt-B0 (ours)	✗	2.18	0.49	1399	11.80	49.25	76.33	52.1	**97.5**	**2.77**
AudioRepInceptionNeXt-B0 (ours)	✓	**2.11**	**0.46**	**2245**	**16.40**	49.25	76.33	52.1	**97.5**	**2.77**
Slow-Fast (baseline) [1]	N/A	26.68	5.55	796	6.90	**52.24**	**78.14**	**54.4**	97.5	2.76
RepLKNet-31T [33]	✗	29.52	7.86	279	0.41	52.22	78.03	54.4	97.5	2.78
RepLKNet-31T [33]	✓	29.32	7.79	295	0.43	52.22	78.03	54.4	97.5	2.78
ResNet50 [46]	N/A	24.13	5.26	915	**7.80**	52.07	77.72	54.1	97.3	2.74
InceptionNeXt-Tiny [48]	N/A	24.20	5.46	682	7.70	50.16	76.28	52.5	97.4	2.75
AudioRepInceptionNeXt-B1(2D)	✗	13.70	3.52	529	0.91	51.26	77.54	53.4	97.5	2.77
AudioRepInceptionNeXt-B1 (2D)	✓	13.19	3.27	666	0.95	51.26	77.54	53.4	97.5	2.77
AudioRepInceptionNeXt-B1 (ours)	✗	11.83	2.62	700	6.10	51.96	77.86	54.0	97.6	2.79
AudioRepInceptionNeXt-B1 (ours)	✓	**11.69**	**2.55**	**1019**	7.50	51.96	77.86	54.0	**97.6**	**2.79**

1) Pretraining: For the pretraining stage, we follow the baseline Slow-Fast model experiment setting [1] and randomly pick a sample of 5.12 seconds from the audio signal followed by feeding it to the Log-Mel filter banks with a window size of 20ms, and a hop length of 10ms. This results in a spectrogram of size 512 × 128, which is given as an input to the model. We follow the training setting described in [1] to train the models using an SGD optimizer for 50 epochs with a momentum of 0.9 and an initial learning rate of 0.01. We drop the learning rate by 0.1 at epochs 30 and 40.

2) Fine-Tuning: We use the same strategy as [1] in the fine-tuning stage. We attach a linear prediction head on top of the VGG-Sound pre-trained backbone model to classify the target classes in different fine-tuning datasets. We froze all the batch normalization layers except the first one in the stem layer and fine-tuned the whole model. We use the same optimizer setting as the pretraining stage, except the initial learning rate is set to 0.001, which is reduced after 20 and 25 epochs by a factor of 0.1. The model is finetuned for 30 epochs.

For the EPIC-KITECHENS-100, we follow the setting described in [1] and randomly pick 2.08 seconds of audio and apply a Log-Mel-Filter bank with a window size of 10ms and a hop length of 5ms. This results in a spectrogram of size 416×128 which is fed to the model as an input. Note that due to the need to downsample the spectrogram by a factor of 32 in our proposed model, adjustments were made to the sampling time, resulting in a slightly longer duration of 2.08 seconds. For EPIC-SOUND, we follow the setting described in [56] and apply a mel-log-filter bank with a window size of 10ms and a hop length of 5ms. However, we randomly pick 2.08 seconds instead of 2 seconds as stated in [56] to account for the downsampling stage in our model. This results in a spectrogram of size 416×128. For KS2 dataset, which has a maximum audio length of 1.023 seconds, we use a similar windows size of 5ms and hop length of 2ms in [56]. This results in a spectrogram of size 512×128. For NSYNTH and Urban8K datasets, following the setup for the VGG sound dataset in [1], we randomly pick a sample of 4.16 seconds from the audio signal and apply a Log-Mel-Filter bank with a window size of 20ms and a hop length of 10ms. This results in a spectrogram of size 416×128.

D. Evaluation Metrics

For the evaluation of the VGG-Sound classification task, we follow the protocol of [1], [2], [39] and report the top-1 accuracy, top-5 accuracy, mean average precision (mAP), area under curve (AUC) and d-prime. For the evaluation of the EPIC-Sounds, KS2, Urban8K, and Nsynth, we report the top-1 and top-5 accuracy. For the EPIC-KITCHENS-100, we follow the evaluation method in [55] and report the top-1 and top-5 accuracy of verb and noun classes. Additionally, we report the top-1 and top-5 accuracy on unseen audio clips in EPIC-KITCHENS-100 to test the generalization ability of the fine-tuned model.

For the measurement of the GPU inference speed (sample/secs), we test all the models on an NVIDIA RTX3090 using a batch size of 32. We first feed 50 batches to warm up the hardware, followed by 50 batches to record the average running time. To measure the CPU inference speed, we conduct the tests on Intel®Xeon®Gold 6226R CPU @ 2.90GHz using a batch size of 1. The tests are performed using a single thread, and we record the average time after the 50 rounds.

E. Effects of Re-parametrization

To verify the effectiveness of the re-parametrization in our proposed AudioRepInceptionNeXt, we perform the comparison against the state-of-the-art (SOTA) methods. The comparison is performed in terms of the parameter size, GFLOPs, throughput, and top-1 accuracy before and after the re-parametrization. We report the results in Table II. One can see that before reparameterization, AudioRepInceptionNeXt-B1 is 12% and 23% slower than the Slow-Fast [1] and ResNet50 [46] respectively, and 60% faster than RepLKNet [33]. We conjecture the low throughput of AudioRepInceptionNeXt is due to its complicated multi-branch design although it incurs lower parameters and theoretical GFLOPs than SlowFast and ResNet50. As discussed in [33]–[35], the large kernel with depthwise convolution increases the memory access costs. Additionally, as mentioned in [35], small operators (i.e., individual convolution (e.g., 1×1) and pooling operations) with multi-branch design are less efficient on GPUs and

TABLE III
TRANSFER LEARNING RESULTS ON EPIC-SOUNDS, KS2, URBAN8K AND NSYNTH. NOTE THAT THE MODELS ARE SEPARATED BY THE MODEL SIZE. THE UPPER THREE ROWS REPRESENT THE SMALL MODEL SIZE, WHILE THE REMAINING ROWS REPRESENT THE LARGE MODEL SIZE.

Model	Param (M)	GFLOPs	EPIC-Sounds		KS2		Urban8K		Nsynth	
			Top-1	Top-5	Top-1	Top-5	Top-1	Top-5	Top-1	Top-5
BCResNet-8 [59]	0.33	1.17	52.91	84.08	95.80	99.19	**84.82**	**98.73**	76.13	95.44
BN-InceptionNetV1 [60]	6.07	1.51	**53.81**	**85.00**	96.84	99.30	82.41	98.39	78.09	94.21
AudioRepInceptionNeXt-B0	**2.02**	**0.37**	53.43	84.77	**97.11**	**99.45**	82.33	98.15	**78.94**	**96.58**
Slow-Fast (baseline) [1]	26.06	4.50	**52.84**	83.12	**97.30**	99.37	81.83	97.13	77.49	96.32
RepLKNet-31T [33]	29.18	6.33	52.79	82.40	97.01	**99.48**	83.40	98.10	**78.48**	96.75
ResNet50 [46]	23.59	4.27	52.57	82.77	97.17	99.37	80.62	97.96	77.56	96.36
InceptionNeXt-Tiny [48]	24.00	4.43	51.24	81.73	96.94	99.34	82.08	97.54	77.24	96.68
AudioRepInceptionNeXt-B1 (2D)	13.05	2.65	51.97	82.35	96.83	99.38	81.93	97.75	77.33	**97.47**
AudioRepInceptionNeXt-B1 (ours)	**11.55**	**2.07**	52.74	**83.22**	97.20	99.39	**83.44**	**98.13**	77.28	97.00

introduce extra kernel launching and synchronization overhead thus reduces the degree of parallelism on GPU. In our AudioRepInceptionNeXt block, the separable kernels of size 1×11, 1×3, 11×1, and 3×1 are relatively small operators compared to 1×21 and 21×1. To address these issues, we employ the re-parametrization technique, as discussed in Section III-C, that results in a network style similar to ResNet50, as shown in Figure 2b. One can see from Table II that the reparametrized version of AudioRepInectionNeXt achieves $1.28\times$, $1.11\times$, and $3.65\times$ improvement in throughput compared to the Slow-Fast, ResNet50, and RepLKNet respectively while maintaining comparable performance. Notably, the re-parametrization technique does not impact the accuracy, and results in lossless compression of the model. In the rest of the sections, we report the evaluation of the reparametrized version of AudioRepInceptionNeXt.

F. Comparison against the CNN-based baselines on VGG Sound pretraining

We compare the performance of AudioRepInceptionNeXt against the CNN-Based baselines that include BCResNet-8 [59], BN-InceptionNet [60], ResNet [46], Slow-Fast model [1], InceptionNeXt [48], RepLKNet [33] and 2D AudioRepInceptionNeXt(without any separable convolution kernel and with branch configurations of 21×21, 11×11 and 3×3 kernels) as shown in Figure 2b (Right). To ensure a fair comparison, we downscaled the original RepLKNet-31B model, with 79M parameters, to the RepLKNet-31T model with 29.3M parameters. This downsizing involved reducing the channel size to 64, 128, 320, and 512 for model stages 1 to 4, respectively. All the evaluation is performed on the VGG-Sound event classification dataset. We report the results in Table II.

Our key findings include: First, the proposed AudioRepInceptionNeXt demonstrates a remarkable reduction in the number of parameters and theoretical GFLOPs while achieving comparable or superior performance compared to other CNN-based methods. For instance, when compared to the multi-stream Slow-Fast model, AudioRepInceptionNeXt-B1 achieves a 56% reduction in parameters and 54% reduction in theoretical GLOPs, with only a slight performance drop of 0.28%. Similarly, in comparison to multi-branch large kernel InceptionNeXt-Tiny, our proposed model achieves a 52% reduction in parameters and a 53% reduction in GFLOPs while attaining a higher accuracy of 1.8%. Furthermore, when compared to 2D large kernel-based RepLKNet-31T, our model achieves comparable accuracy with a difference of only 0.26% while saving 71.05% of parameters and 68% of GFLOPs. Additionally, compared to the 2D version of the proposed AudioRepInceptionNeXt-B1, our model achieves a 0.7% higher accuracy while saving 11% of model parameters and 22% of GFLOPs. This result is consistent with the findings in [1], [32], demonstrating that the model with separable kernels performs better by enabling the extraction of temporal and frequency information independently. Second, our model demonstrates the fastest inference speed on GPU compared to other CNN-based baselines, aligning with the theoretical GFLOPs. The proposed AudioRepInceptionNeXt-B0 obtains 2245 fps (vs. 2031 fps and 492 fps obtained by BN-InceptionNetV1 and BCResNet-8, respectively), while AudioRepIncetpionNeXt-B1 obtains 1019 fps (vs.796 fps obtained by Slow-Fast). Third, the proposed AudioRepInceptionNeXt-B1 achieves a higher inference speed on the CPU compared to the Slow-Fast model, RepLKNet-31T, AudioRepInceptionNeXt-B1 2D, and BN-InceptionNetV1, while maintaining a comparable performance with Slow-Fast and RepLKNet-31T. We also note that the CPU inference speed of AudioRepInceptionNeXt-B1 is comparable to ResNet50 and InceptionNeXt-Tiny while surpassing InceptionNeXt-Tiny in terms of accuracy and achieving competitive accuracy compared to ResNet50. When comparing the inference speeds of ResNet50 and AudioRepInceptionNeXt-B1 on CPU and GPU, we conjecture that the difference in the inference speed among these models can be attributed to the optimized implementation of depthwise convolution on GPU compared to CPU, as discussed in [61]. Additionally, the disparity in arithmetic intensity (the ratio of compute to memory operations), as mentioned in Section IV-E, also contributes to the observed variations in speed on GPU and CPU. These findings highlight the superior performance of AudioRepInceptionNeXt in terms of parameter efficiency, theoretical GFLOPs, and inference speed compared to other CNN-based baselines.

G. Performance on Downstream task datasets

To verify the conclusion from Section IV-F, we conduct transfer learning experiments on multiple datasets: Speech Command V2 [57], UrbanSound8K [62], EPIC-KITCHEN-100 [55], NSynth [4] and EPIC-Sound [56]. As shown in Table

TABLE IV
TRANSFER LEARNING RESULTS ON EPIC-KITCHENS-100. NOTE THAT THE MODELS ARE SEPARATED BY THE MODEL SIZE. THE UPPER THREE ROWS REPRESENT THE SMALL MODEL SIZE, WHILE THE REMAINING ROWS REPRESENT THE LARGE MODEL SIZE.

Model	Param (M)	GFLOPs	Overall Top-1 Accuracy Verb	Noun	Action	Overall Top-5 Accuracy Verb	Noun	Action	Unseen Participants Top-1 Accuracy Verb	Noun	Action
BCResNet-8 [59]	0.41	1.17	42.26	18.14	1.99	77.72	42.00	2.19	37.18	17.84	2.44
BN-inceptionNetV1 [60]	6.44	1.51	44.68	**21.03**	12.90	79.94	46.36	27.77	38.30	17.27	9.67
AudioRepInceptionNeXt-B0	**2.14**	**0.37**	**46.34**	20.71	**13.51**	**80.52**	**46.75**	**29.77**	**40.85**	**17.28**	**9.86**
Slow-Fast (baseline)	26.88	4.33	46.86	**22.98**	15.52	80.12	47.58	30.17	39.62	17.28	**10.52**
RepLKNet-31T [33]	29.36	6.33	47.33	23.46	**16.12**	80.25	47.95	30.75	40.00	**17.37**	9.48
ResNet50 [46]	24.32	4.28	46.03	22.79	15.21	**80.71**	47.83	30.06	**40.84**	16.24	9.76
InceptionNeXt-Tiny [48]	24.27	4.43	44.72	21.80	14.07	79.27	45.93	28.18	38.77	15.96	9.10
AudioRepInceptionNeXt-B1 (2D)	13.23	2.65	46.89	22.37	15.01	79.95	**48.12**	31.09	40.46	16.99	9.57
AudioRepInceptionNeXt-B1	**11.73**	**2.07**	**47.57**	22.14	15.42	80.45	48.06	**31.17**	39.62	17.00	9.57

TABLE V
MODEL SIZE AND INFERENCE TIME COMPARISON BETWEEN CNN-BASED METHODS AND AUDIOREPINCEPTIONNEXT ON THE MOBILE DEVICE. LOWER INFERENCE TIME IS BETTER. NOTE THAT THE MODELS ARE SEPARATED BY THE MODEL SIZE. THE UPPER THREE ROWS REPRESENT THE SMALL MODEL SIZE, WHILE THE REMAINING ROWS REPRESENT THE LARGE MODEL SIZE.

Model	Param (M)	GFLOPs	Model Size (MB)	Inference Time (ms)
BCResNet-8 [59]	**0.33**	1.17	**1.6**	227
BN-InceptionNetV1 [60]	6.35	1.86	25	114
AudioRepInceptionNeXt-B0 (ours)	2.11	**0.46**	9	**59**
Slow-Fast (baseline) [1]	26.68	5.55	106	317
RepLKNet-31T [33]	29.32	7.79	117	2136
ResNet50 [46]	24.13	5.26	96	310
InceptionNeXt-Tiny [48]	24.20	5.46	97	357
AudioRepInceptionNeXt-B1 (2D)	13.19	3.27	53	615
AudioRepInceptionNeXt-B1 (ours)	**11.69**	**2.55**	**47**	**232**

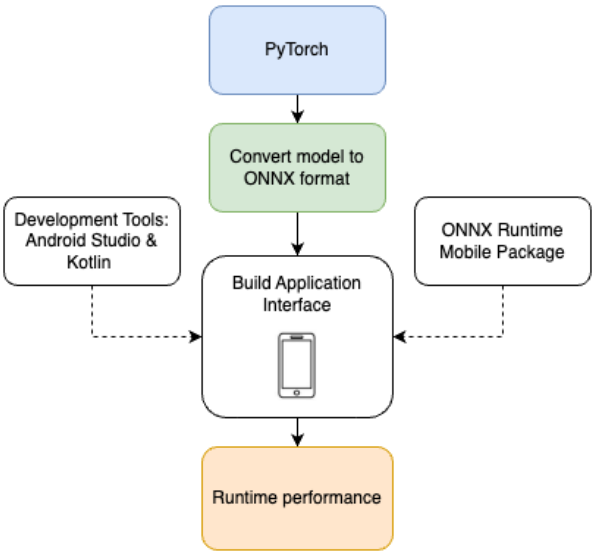


Fig. 4. Mobile runtime application development flow by using the ONNX Runtime library.

III and Table IV, our proposed model achieves comparable performance to the multi-stream Slow-Fast model in terms of accuracy on four transfer learning datasets, while saving 56% and 54% of parameters and GFLOPs, respectively. Moreover, when compared to the Urban8K dataset our model outperforms the Slow-Fast model by 1.61% in terms of top-1 accuracy. Additionally, when compared to other CNN-based methods, our methods achieve comparable or superior performance in terms of accuracy, while having the lowest parameters count and computational GFLOPs. These results indicate that our model can learn the rich representations that are applicable to different domains and are robust when transferred to various audio understanding tasks. These findings are also aligned with the results obtained from the pretraining on the VGG Sound dataset, as discussed in section IV-F.

H. Implementation on Mobile Devices

To evaluate the inference speed on the mobile device, we deploy all models on an Android mobile platform. The implementation procedure is summarized in Fig.4. We first convert the PyTorch models to the Open Neural Network Exchange (ONNX) format [63]. It is an open-source machine-independent format compatible with different hardware, drivers, and operating systems. We then utilize the Android development tools, specifically Android Studio and Kotlin, to build an application interface for conducting the speed evaluation. To run the ONNX models on Android mobile devices, we leverage the ONNX Runtime mobile package. It provides a lightweight and optimized runtime specifically designed for mobile platforms. During the evaluation, we run all models on Redmi Note 9 Pro smartphone with Snapdragon 720G SoC with floating point 32. Specifically, we feed an input with a batch size of 1 and record the average inference time by processing 50 batches.

I. Runtime Performance on Mobile Devices

As shown in Table V, our model demonstrates superior performance in terms of inference speed and model size on mobile devices compared to other state-of-the-art CNNs. Notably, the

TABLE VI
ABLATION STUDY ON THE DESIGN OF MULTI-BRANCH LARGE KERNEL.

Structure	3×3	11×11	21×21	31×31	Identity	Inverted Bottleneck	Param (M)		GFLOPs		Throughput		Top-1	Top-5	mAP	AUC	d-prime
							Before Rep.	After Rep.	Before Rep.	After Rep.	Before Rep.	After Rep.					
s1	✓	✗	✗	✗	✓	✓	11.56	11.54	2.49	2.48	1118	1175	51.40	77.79	53.82	97.43	2.75
s2	✗	✓	✗	✗	✓	✓	11.62	11.60	2.52	2.51	1048	1101	51.55	77.46	53.76	97.50	2.77
s3	✗	✗	✓	✗	✓	✓	11.69	11.68	2.55	2.55	971	1019	51.36	77.83	53.54	97.61	2.79
s4	✓	✗	✓	✗	✓	✓	11.73	11.69	2.57	2.55	833	1019	51.60	77.74	53.87	97.55	2.78
s5	✗	✓	✓	✗	✓	✓	11.79	11.69	2.60	2.55	794	1019	51.64	77.68	53.66	97.55	2.78
s6	✓	✓	✓	✗	✓	✓	11.83	11.69	2.62	2.55	701	1019	**51.96**	**77.86**	53.97	97.58	2.79
s7	✓	✓	✓	✓	✓	✓	12.08	11.69	2.74	2.55	554	1001	51.49	77.77	**54.23**	97.58	2.79
s8	✓	✓	✓	✗	✗	✓	11.83	11.69	2.62	2.55	727	1076	51.29	77.44	53.68	**97.62**	**2.80**
s9	✓	✓	✓	✗	✓	✗	3.02	2.85	0.73	0.65	1011	1803	51.02	77.41	53.67	97.61	2.79

AudioRepInceptionNeXt-B1 model outperforms the classical ResNet50 by reducing the inference time and model size by 25% and 50%, respectively. In comparison to the recent multi-branch large kernel InceptionNeXt-Tiny, our model achieves a 35% faster inference speed and 51% smaller model size. When compared to RepLKNet-31T, a 2D large kernel-based model, our proposed model exhibits substantial improvements, saving 89% of the inference time and reducing the model size by 60%. Moreover, our method surpasses the multi-stream Slow-Fast model, achieving 27% lower inference time and 56% smaller model size. The results clearly demonstrate the efficiency of our model in terms of both inference speed and model size on mobile devices, showcasing its superiority over existing state-of-the-art CNNs.

V. Ablation Studies

We conduct a series of ablation studies on AudioRepInceptionNeXt to verify the significance of multi-branch design and the usage of large kernels. To save the compute, we only conduct ablation studies on the VGG-Sound dataset and AudioRepInceptionNeXt-B1 architecture. Specifically, we first ablate some branches with different kernel sizes and then observe the performance changes. We then compare the AudioRepInceptionNeXt block to a counterpart with a block without a shortcut path. Concretely, we remove the identity shortcut for both horizontal and vertical multi-scale kernels. Note that we follow the same training setting as mentioned in section IV. We report the results of such an ablation study in terms of the number of parameters, GFLOPs, throughput, and accuracy before and after re-parameterization in Table VI.

As shown in Table VI, for the single branch setting (i.e., structure s1 to s3), there is no significant accuracy improvement when we enlarge the kernel size from 3 to 11. Meanwhile, there is 0.19% performance degradation when we further enlarge the kernel size from 11 to 21. We conjecture that the model overlooks the local details when we enlarge the respective field of the kernel. To verify it, we introduce the multi-branch setting (i.e., structure s4 to s6) with kernel sizes 11 and 3 in our ablation studies. The results demonstrate that all the multi-branch models can lift the accuracy above 51.55% and outperform the single-branch models. Comparing the triple branch (i.e., structure s6) to the dual branch (i.e., structure s4 and s5), we note that removing any single branch degrades the performance, suggesting that all the branches are indispensable. However, when we introduced additional branches with a kernel size of 31×31, the accuracy started to degrade by 0.4%.

To verify the importance of the identity shortcut in the parallel multi-scale horizontal and vertical kernel, we remove all the identity shortcut layers and form a new structure s8 as shown in table VI. Compared to the full structure s6, the accuracy of s8 is dropped by 0.67%. This indicates the importance of identity shortcuts in the AudioRepInceptionNeXt block.

To further verify the importance of the inverted bottleneck layer, we remove the 1×1 channel expansion layer and form a new structure s9 as shown in table VI. Compared to the full structure s6, the accuracy of s9 is dropped by 0.94%. This demonstrates the importance of the inverted bottleneck.

VI. Conclusion

In this study, we address the issue of computational and memory inefficiency in the multi-stream and multi-branch convolutional neural networks (CNNs). To alleviate these problems, a simple single-stream model was proposed which employs a parallel multi-scale separable kernel design, effectively reducing the number of parameter counts and GFLOPs by 50% during the training. In order to eliminate the side effects arising from multi-scale kernel design, such as slow inference speed and frequent memory access, we utilize a reparameterization technique during the inference. Moreover, we adopt depthwise separable kernels and an inverted bottleneck design to address the inefficiencies associated with the large-scale kernels in the parallel branches of the multi-scale kernel. Our experimental results show that the proposed AudioRepInceptionNeXt achieves a favorable trade-off between parameter size and computational time while maintaining comparable or superior performance in relation to the baseline Slow-Fast model and AudioRepInceptionNeXt (2D). Additionally, our findings demonstrate that the proposed model is a robust learner capable of achieving better or comparable performance compared to the SlowFast model across various transfer learning datasets. This study provides valuable insights for researchers and practitioners in the field of deep learning who seek to enhance the computational efficiency of audio classification models.

References

[1] E. Kazakos, A. Nagrani, A. Zisserman, and D. Damen, "Slow-fast auditory streams for audio recognition," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 855–859. 1, 2, 3, 5, 7, 8, 9, 10

[2] S. Hershey, S. Chaudhuri, D. P. Ellis, J. F. Gemmeke, A. Jansen, R. C. Moore, M. Plakal, D. Platt, R. A. Saurous, B. Seybold *et al.*, "Cnn architectures for large-scale audio classification," in *2017 ieee international conference on acoustics, speech and signal processing (icassp)*. IEEE, 2017, pp. 131–135. 1, 3, 8

[3] S. Verbitskiy, V. Berikov, and V. Vyshegorodtsev, "Eranns: Efficient residual audio neural networks for audio pattern recognition," *Pattern Recognition Letters*, vol. 161, pp. 38–44, 2022. 1, 3

[4] J. Engel, C. Resnick, A. Roberts, S. Dieleman, M. Norouzi, D. Eck, and K. Simonyan, "Neural audio synthesis of musical notes with wavenet autoencoders," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1068–1077. 1, 7, 9

[5] S. Allamy and A. L. Koerich, "1d cnn architectures for music genre classification," in *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2021, pp. 01–07. 1, 3

[6] I. Lezhenin, N. Bogach, and E. Pyshkin, "Urban sound classification using long short-term memory neural network," in *2019 federated conference on computer science and information systems (FedCSIS)*. IEEE, 2019, pp. 57–60. 1

[7] G. Duan, S. Zhang, M. Lu, C. Okinda, M. Shen, and T. Norton, "Short-term feeding behaviour sound classification method for sheep using lstm networks," *International Journal of Agricultural and Biological Engineering*, vol. 14, no. 2, pp. 43–54, 2021. 1

[8] D. Utebayeva, A. Almagambetov, M. Alduraibi, Y. Temirgaliyev, L. Ilipbayeva, and S. Marxuly, "Multi-label uav sound classification using stacked bidirectional lstm," in *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2020, pp. 453–458. 1

[9] J. K. Das, A. Ghosh, A. K. Pal, S. Dutta, and A. Chakrabarty, "Urban sound classification using convolutional neural network and long short term memory based on multiple features," in *2020 Fourth International Conference On Intelligent Computing in Data Sciences (ICDS)*. IEEE, 2020, pp. 1–9. 1

[10] Y. Gong, Y.-A. Chung, and J. Glass, "Ast: Audio spectrogram transformer," *arXiv preprint arXiv:2104.01778*, 2021. 1, 7

[11] Y. Gong, C.-I. Lai, Y.-A. Chung, and J. Glass, "Ssast: Self-supervised audio spectrogram transformer," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 36, no. 10, 2022, pp. 10 699–10 709. 1, 7

[12] K. Chen, X. Du, B. Zhu, Z. Ma, T. Berg-Kirkpatrick, and S. Dubnov, "Hts-at: A hierarchical token-semantic audio transformer for sound classification and detection," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 646–650. 1, 7

[13] A. Saeed, D. Grangier, and N. Zeghidour, "Contrastive learning of general-purpose audio representations," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 3875–3879. 1

[14] D. Niizumi, D. Takeuchi, Y. Ohishi, N. Harada, and K. Kashino, "Byol for audio: Self-supervised learning for general-purpose audio representation," in *2021 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021, pp. 1–8. 1

[15] L. Wang and A. v. d. Oord, "Multi-format contrastive learning of audio representations," *arXiv preprint arXiv:2103.06508*, 2021. 1

[16] S. Suh, W. Lim, S. Park, and Y. Jeong, "Acoustic scene classification using specaugment and convolutional neural network with inception modules," *Proceedings of the DCASE2019 Challenge, New York, NY, USA*, pp. 25–26, 2019. 1

[17] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "Specaugment: A simple data augmentation method for automatic speech recognition," *arXiv preprint arXiv:1904.08779*, 2019. 1, 7

[18] A. Baevski, Y. Zhou, A. Mohamed, and M. Auli, "wav2vec 2.0: A framework for self-supervised learning of speech representations," *Advances in neural information processing systems*, vol. 33, pp. 12 449–12 460, 2020. 1, 3

[19] L. Ford, H. Tang, F. Grondin, and J. R. Glass, "A deep residual network for large-scale acoustic scene analysis." in *InterSpeech*, 2019, pp. 2568–2572. 1

[20] F. Schmid, K. Koutini, and G. Widmer, "Efficient large-scale audio tagging via transformer-to-cnn knowledge distillation," *arXiv preprint arXiv:2211.04772*, 2022. 1

[21] Y. Gong, Y.-A. Chung, and J. Glass, "Psla: Improving audio tagging with pretraining, sampling, labeling, and aggregation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 29, pp. 3292–3306, 2021. 1, 3

[22] L. Wang, P. Luc, Y. Wu, A. Recasens, L. Smaira, A. Brock, A. Jaegle, J.-B. Alayrac, S. Dieleman, J. Carreira *et al.*, "Towards learning universal audio representations," in *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2022, pp. 4593–4597. 1

[23] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, "Panns: Large-scale pretrained audio neural networks for audio pattern recognition," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 2880–2894, 2020. 1, 3

[24] A. Mesaros, T. Heittola, T. Virtanen, and M. D. Plumbley, "Sound event detection: A tutorial," *IEEE Signal Processing Magazine*, vol. 38, no. 5, pp. 67–83, 2021. 1

[25] K. Ahuja, A. Kong, M. Goel, and C. Harrison, "Direction-of-voice (dov) estimation for intuitive speech interaction with smart devices ecosystems." in *UIST*, 2020, pp. 1121–1131. 1

[26] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018. 1

[27] A. Nagrani, J. S. Chung, and A. Zisserman, "Voxceleb: a large-scale speaker identification dataset," *arXiv preprint arXiv:1706.08612*, 2017. 1

[28] K. Chen, C.-i. Wang, T. Berg-Kirkpatrick, and S. Dubnov, "Music sketchnet: Controllable music generation via factorized representations of pitch and rhythm," *arXiv preprint arXiv:2008.01291*, 2020. 1

[29] Y. Gao, J. Fernandez-Marques, T. Parcollet, A. Mehrotra, and N. D. Lane, "Federated self-supervised speech representations: Are we there yet?" *arXiv preprint arXiv:2204.02804*, 2022. 1

[30] Y. Gaol, J. Fernandez-Marques, T. Parcollet, P. P. de Gusmao, and N. D. Lane, "Match to win: Analysing sequences lengths for efficient self-supervised learning in speech and audio," in *2022 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 2023, pp. 115–122. 1

[31] K. W. Lau, Y. A. U. Rehman, Y. Xie, and L. Ma, "Audioinceptionnext: Tcl ai lab submission to epic-sound audio-based-interaction-recognition challenge 2023," 2023. 1, 2

[32] F. Xiao, Y. J. Lee, K. Grauman, J. Malik, and C. Feichtenhofer, "Audiovisual slowfast networks for video recognition," *arXiv preprint arXiv:2001.08740*, 2020. 1, 5, 9

[33] X. Ding, X. Zhang, J. Han, and G. Ding, "Scaling up your kernels to 31x31: Revisiting large kernel design in cnns," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 963–11 975. 2, 3, 6, 8, 9, 10

[34] X. Ding, X. Zhang, N. Ma, J. Han, G. Ding, and J. Sun, "Repvgg: Making vgg-style convnets great again," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2021, pp. 13 733–13 742. 2, 3, 8

[35] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "Shufflenet v2: Practical guidelines for efficient cnn architecture design," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 116–131. 2, 3, 8

[36] M.-H. Guo, C.-Z. Lu, Z.-N. Liu, M.-M. Cheng, and S.-M. Hu, "Visual attention network," *arXiv preprint arXiv:2202.09741*, 2022. 2

[37] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 11 976–11 986. 2, 5

[38] X. Ding, X. Zhang, J. Han, and G. Ding, "Diverse branch block: Building a convolution as an inception-like unit," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 886–10 895. 3

[39] H. Chen, W. Xie, A. Vedaldi, and A. Zisserman, "Vggsound: A large-scale audio-visual dataset," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 721–725. 3, 6, 7, 8

[40] J. Salamon and J. P. Bello, "Deep convolutional neural networks and data augmentation for environmental sound classification," *IEEE Signal processing letters*, vol. 24, no. 3, pp. 279–283, 2017. 3

[41] Y. Aytar, C. Vondrick, and A. Torralba, "Soundnet: Learning sound representations from unlabeled video," *Advances in neural information processing systems*, vol. 29, 2016. 3

[42] I. Martín-Morató, F. Paissan, A. Ancilotto, T. Heittola, A. Mesaros, E. Farella, A. Brutti, and T. Virtanen, "Low-complexity acoustic scene classification in dcase 2022 challenge," *arXiv preprint arXiv:2206.03835*, 2022. 3

[43] S. Schneider, A. Baevski, R. Collobert, and M. Auli, "wav2vec: Unsupervised pre-training for speech recognition," *arXiv preprint arXiv:1904.05862*, 2019. 3

[44] F. Li, M. Liu, Y. Zhao, L. Kong, L. Dong, X. Liu, and M. Hui, "Feature extraction and classification of heart sound using 1d convolutional neural networks," *EURASIP Journal on Advances in Signal Processing*, vol. 2019, no. 1, pp. 1–11, 2019. 3

[45] F. Schmid, K. Koutini, and G. Widmer, "Efficient large-scale audio tagging via transformer-to-cnn knowledge distillation," in *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2023, pp. 1–5. 3

[46] A. Jansen, M. Plakal, R. Pandya, D. P. Ellis, S. Hershey, J. Liu, R. C. Moore, and R. A. Saurous, "Unsupervised learning of semantic audio representations," in *2018 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2018, pp. 126–130. 3, 8, 9, 10

[47] X. Li, V. Chebiyyam, and K. Kirchhoff, "Multi-stream network with temporal attention for environmental sound classification," *arXiv preprint arXiv:1901.08608*, 2019. 3

[48] W. Yu, P. Zhou, S. Yan, and X. Wang, "Inceptionnext: When inception meets convnext," *arXiv preprint arXiv:2303.16900*, 2023. 3, 8, 9, 10

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778. 5

[50] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826. 5

[51] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017. 5

[52] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 4510–4520. 5

[53] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1492–1500. 5

[54] A. Veit, M. J. Wilber, and S. Belongie, "Residual networks behave like ensembles of relatively shallow networks," *Advances in neural information processing systems*, vol. 29, 2016. 6

[55] D. Damen, H. Doughty, G. M. Farinella, A. Furnari, E. Kazakos, J. Ma, D. Moltisanti, J. Munro, T. Perrett, W. Price *et al.*, "Rescaling egocentric vision," *arXiv preprint arXiv:2006.13256*, 2020. 7, 8, 9

[56] J. Huh, J. Chalk, E. Kazakos, D. Damen, and A. Zisserman, "Epic-sounds: A large-scale dataset of actions that sound," *arXiv preprint arXiv:2302.00646*, 2023. 7, 8, 9

[57] P. Warden, "Speech Commands: A Dataset for Limited-Vocabulary Speech Recognition," *ArXiv e-prints*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.03209 7, 9

[58] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *22nd ACM International Conference on Multimedia (ACM-MM'14)*, Orlando, FL, USA, Nov. 2014, pp. 1041–1044. 7

[59] B. Kim, S. Chang, J. Lee, and D. Sung, "Broadcasted residual learning for efficient keyword spotting," *arXiv preprint arXiv:2106.04140*, 2021. 8, 9, 10

[60] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*. pmlr, 2015, pp. 448–456. 8, 9, 10

[61] G. Lu, W. Zhang, and Z. Wang, "Optimizing depthwise separable convolution operations on gpus," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 1, pp. 70–87, 2021. 9

[62] J. Salamon, C. Jacoby, and J. P. Bello, "A dataset and taxonomy for urban sound research," in *Proceedings of the 22nd ACM international conference on Multimedia*, 2014, pp. 1041–1044. 9

[63] J. Bai, F. Lu, K. Zhang *et al.*, "Onnx: Open neural network exchange," 2019. 10