ADASTRIDE: USING ADAPTIVE STRIDES IN SEQUENTIAL DATA FOR EFFECTIVE DOWNSAMPLING

Anonymous authors
Paper under double-blind review

ABSTRACT

The downsampling layer has been one of the most commonly used deep learning (DL) components in sequential data processing due to its several advantages. First, it improves the generalization performance of networks by acting as an information bottleneck, where it extracts task-relevant features and discards others. Second, it reduces data resolution allowing CNN layers to have larger receptive fields with smaller kernel sizes. Third, the reduced data resolution facilitates the use of Transformer networks in case of high-resolution data. Accordingly, there have been many studies on downsampling methods, but they have a limitation in that they apply the same downsampling ratio across a data instance. Using the same downsampling ratio uniformly for an entire data instance does not reflect the fact that the task-relevant information is not uniformly distributed in real data. In this paper, we introduce AdaStride, a downsampling method that can apply adaptively varying downsampling ratios across a sequential data instance given an overall downsampling ratio. Specifically, AdaStride learns to deploy adaptive strides in a sequential data instance. Therefore, it can preserve more information from task-relevant parts of a data instance by using smaller strides for those parts and larger strides for less relevant parts. To achieve this, we propose a novel training method called vector positioning that rearranges each time step of an input on a one-dimensional line segment without reordering, which is used to build an alignment matrix for the downsampling. In experiments conducted on three different tasks of audio classification, automatic speech recognition, and discrete representation learning, AdaStride outperforms other widely used standard downsampling methods showing its generality and effectiveness. In addition, we analyze how our AdaStride learns the effective adaptive strides to improve its performance in the tasks.

1 INTRODUCTION

Recently, deep learning (DL) has achieved remarkable performance in various machine learning domains such as image classification (Krizhevsky et al., 2012; He et al., 2016a), machine translation (Bahdanau et al., 2015; Vaswani et al., 2017), audio classification (Yoon et al., 2019; Li et al., 2019), and speech recognition (Chan et al., 2015; Gulati et al., 2020; Kim et al., 2022). This is because many DL architectures such as CNN (Fukushima & Miyake, 1982; LeCun et al., 1989), RNN (Rumelhart et al., 1985; Hochreiter & Schmidhuber, 1997) and Transformer (Vaswani et al., 2017) can be easily employed for various types of input and output.

Especially, downsampling layers have brought many benefits when they are used in combination with other DL layers in many sequential processing tasks. For example, in many classification networks (Li et al., 2019; Ma et al., 2021), the downsampling layer was used with CNN layers where it gradually reduces the data resolution while providing several benefits: (1) it improves the generalization performance of the networks by acting as an information bottleneck that preserves task-relevant information and discards other trivial information (Li & Liu, 2019); (2) it reduces the amount of computation because the reduced resolution allows intermediate CNN layers to have virtually larger receptive fields with smaller kernel sizes. Other than CNN, there have been many studies reporting remarkable results and even the state-of-the-art performance by using downsampling layers and Transformer layers (Dhariwal et al., 2020; Gulati et al., 2020; Kim et al., 2022; Karita et al., 2019; Synnaeve et al., 2019; Collobert et al., 2020) together. In these studies, the downsampling

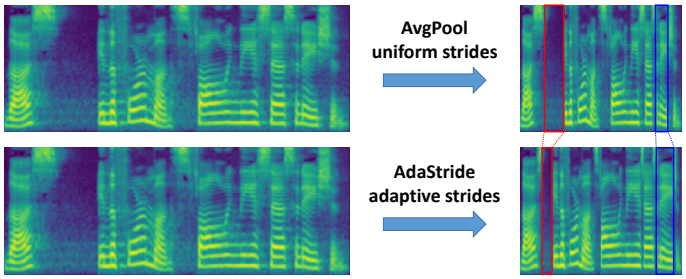


Figure 1: Examples of two different downsampling methods, where they reduce the length of melspectrogram in half: (1) average pooling with uniform strides; (2) AdaStride downsampling that uses adaptive strides. In the case of AdaStride, we apply the downsampling, which is actually learned on hidden representations, to the raw melspectrogram for better understandability.

layer was used to alleviate the problem of quadratically increasing memory and computational cost of Transformer, which has hindered Transformer from being used in case of high-resolution data.

Due to the advantages of the downsampling layers, there have been many studies on downsampling to preserve more task-relevant information or to discard task-interfering information. Radford et al. (2016) replaced the max- or avg- pooling with strided convolutions to improve the performance by allowing the networks to learn their own downsampling. Zhang (2019) pointed out that aliasing can occur in downsampling because the reduced sampling rate decreases the Nyquist frequency (Nyquist, 1928). Therefore, Zhang (2019) proposed to apply low-pass filtering before downsampling to remove the high-frequency components over the Nyquist frequency. Meanwhile, Rippel et al. (2015) proposed spectral pooling that performs downsampling in the frequency domain. Specifically, inspired by the fact that the power spectrum of images mostly consists of lower frequencies, it removes the high-frequency components in the frequency domain. Furthermore, Riad et al. (2022) proposed DiffStride, which improves the performance and narrows down the architecture search space by making the stride factor of the spectral pooling learnable. However, there have been no studies about using varying downsampling ratios across a data instance to fully utilize the given overall downsampling ratio.

In this paper, we introduce AdaStride, a novel downsampling method that applies adaptively varying downsampling ratios in a sequential data sample given an overall downsampling ratio. To be specific, our AdaStride learns to deploy adaptive strides in a sequential data sample. Figure 1 illustrates the usefulness of deploying adaptive strides in a data instance. The upper part of Figure 1 shows average pooling using uniform strides, which inevitably uniformly reduces the data resolution in half for every part of the data. On the contrary, as shown in the lower part of Figure 1, deploying adaptive strides allows for preserving more information for task-relevant parts by using smaller strides for that parts and larger strides for less important parts. To achieve this, we propose a novel training method called vector positioning (VP), which rearranges each time step of the input signal on a one-dimensional line segment without reordering. Based on VP, AdaStride constructs an alignment matrix and performs downsampling based on the matrix. In addition, we also introduce a variant of AdaStride called AdaStride-F, which speeds up the computation by replacing the matrix multiplication in the downsampling with a faster `scatter_add` operation. This reduces the training time of AdaStride by around 13% while showing similar performance compared to AdaStride.

To evaluate AdaStride, we compare our method with widely used downsampling methods in three different tasks: audio classification, automatic speech recognition (ASR), and discrete representation learning based on VQ-VAE (Van Den Oord et al., 2017). In audio classification, AdaStride outperforms all the standard downsampling methods including DiffStride on three of four datasets without hyperparameter tuning. Also, we identify that it is possible to obtain better performance of AdaStride by adjusting the trade-off between information loss and aliasing depending on the dataset. Indeed, AdaStride ranks first on all of the datasets when we optimize the trade-off independently by tuning a hyperparameter according to the dataset. Furthermore, our AdaStride achieves significant performance improvement over the strided convolution, even when the increase in memory usage of AdaStride is minimized. On other tasks, we show that AdaStride also outperforms all the other downsampling methods by learning effective adaptive strides, and we provide analyses of how the adaptive strides are learned in the tasks.

2 BACKGROUND

In this section, we present the background for developing a better understanding of our work. First, we demonstrate how the amount of information that can be preserved in downsampling is bounded, mainly focusing on the relationship between downsampling and aliasing. Then, we will explain spectral pooling based downsampling methods that will appear in comparative experiments, which will also help understand the motivation of our work.

2.1 DOWNSAMPLING AND ALIASING

Zhang et al. (2019) showed that commonly used downsampling methods such as max-pooling, average-pooling, or strided convolution can be seen as a combination of '*densely refining an input*' and '*naively resampling the signal*'. For example, let's assume that we perform max-pooling with a window size of w and a stride of s. Then, it can be decomposed into the following two steps: (1) applying a max operator of a window size of w to the input signal with a stride of 1; (2) naively resampling the refined input signal with a sampling rate reduced by $1/s$.

In the paper, it was pointed out that aliasing can occur in step (2), and it hurts the shift-invariance and shift-equivariance of networks. Therefore, it proposed to apply low-pass filtering to the signal before step (2), and it actually improved the shift-invariance and shift-equivariance. Furthermore, the low-pass filtering surprisingly even improved the classification performance of networks despite of the information loss for the high-frequency components. It implies that, in downsampling, the components above the Nyquist frequency rather disturb the networks than convey task-relevant information. In other words, it is essential to use a lower downsampling ratio to raise the level at which frequency components become meaningful.

2.2 SPECTRAL POOLING AND DIFFSTRIDE

Rippel et al. (2015) proposed spectral pooling that performs downsampling in the frequency domain. Let $\mathbf{x} = \{x_0, ..., x_{T-1}\} \in \mathbb{R}^T$ be the input and s be a stride factor. Then, spectral pooling first sends $\mathbf{x}$ to the frequency domain based on Discrete Fourier Transform (DFT) as $\mathbf{y} = \mathcal{F}(\mathbf{x}) \in \mathbb{C}^{\lfloor \frac{T}{2} \rfloor + 1}$ as follows:

$$y_n = \sum_{t=0}^{T-1} x_t \cdot e^{-i(\frac{2\pi n}{T})t}, \qquad 0 \leq n \leq \left\lfloor \frac{T}{2} \right\rfloor. \tag{1}$$

Then, $\mathbf{y}$ is cropped by maintaining only the lower $\lceil \frac{T}{2s} \rceil + 1$ frequency components, and it is brought back to the original domain through the inverse DFT as $\hat{\mathbf{x}} = \mathcal{F}^{-1}(\mathsf{crop}(\mathbf{y}))$. Inspired by the fact that the power spectrum of natural images mostly consists of low-frequency components, spectral pooling could improve the network performance, and the aliasing problem is naturally resolved here.

Furthermore, Riad et al. (2022) proposed DiffStride that rather applies soft mask $\mathbf{m} \in \mathbb{R}^{\lfloor \frac{T}{2} \rfloor + 1}$ to $\mathbf{y}$ than directly crops $\mathbf{y}$ as follow:

$$m_n = \min\left[\max\left[\frac{1}{R}(R + \frac{T}{2s} + 1 - n),\ 0\right],\ 1\right], \qquad \hat{\mathbf{y}} = \mathbf{m} \odot \mathbf{y}, \tag{2}$$

where R is a hyperparameter and $\odot$ denotes element-wise multiplication. Then, DiffStride crops $\hat{\mathbf{y}}$ by maintaining only the non-zero frequency components and the downsampled signal is obtained as $\hat{\mathbf{x}} = \mathcal{F}^{-1}(\mathsf{crop}(\hat{\mathbf{y}}))$. By doing so, it allowed the model to learn the stride factor s through gradient descent algorithm, resulting in improved classification performance and smaller architecture search space. However, it is also the same for these two spectral pooling methods that the downsampling ratio determines the level at which frequency components become meaningful.

3 METHODOLOGY

In this section, we introduce AdaStride, a novel downsampling method that can adaptively apply different downsampling ratios to different sections of a sequential input based on vector positioning process. Also, we introduce a variant of AdaStride called AdaStride-F, which is an efficient version of AdaStride that performs downsampling based on `scatter_add` operation.

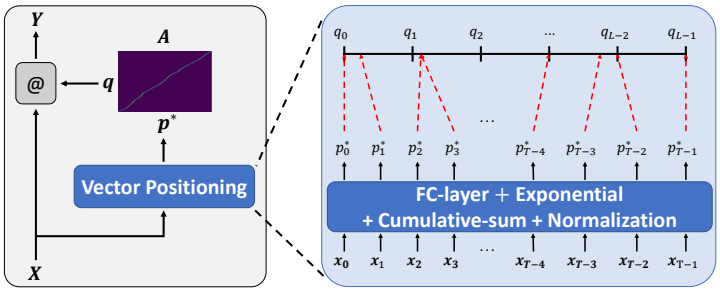


Figure 2: An abstract diagram that illustrates how AdaStride downsampling is performed. The alignment matrix used for downsampling is calculated based on the vector positioning process.

3.1 AdaStride

Let $\mathbf{X} = \{\mathbf{x}_0, ..., \mathbf{x}_{T-1}\} \in \mathbb{R}^{T \times D}$ be the input of AdaStride and s be a stride factor. Then, AdaStride reduces the time steps of $\mathbf{X}$ by the factor of s and outputs $\mathbf{Y} = \{\mathbf{y}_0, ..., \mathbf{y}_{L-1}\} \in \mathbb{R}^{L \times D}$ where $L = \lceil T/s \rceil$. To reduce the time steps of input, we first define a one-dimensional line segment represented by a list of indices $\mathbf{q} = [0, 1, ..., L-1] \in \mathbb{R}^L$ (Miao et al., 2021) as shown in Figure 2. Then, each input vector $\mathbf{x_t}$ is positioned on the line segment following the vector positioning (VP) process. First, we obtain relevant displacements of the input vectors, $\Delta\mathbf{p} = \{\Delta p_0, ..., \Delta p_{T-1}\} \in \mathbb{R}^T$. Here, each Δp_t is calculated by passing each $\mathbf{x_t}$ through a position-wise fully-connected layer followed by a exponential function. Then, the positions of $\mathbf{x_t}$s are calculated using the cumulative summation operation, and they are lastly normalized to be positioned between $[0,\ L-1]$ as follows:

$$p_t = \sum_{n=0}^{t} \Delta p_n, \qquad p_t^* = \frac{p_t - p_0}{p_{T-1} - p_0} * (L-1). \tag{3}$$

In this way, we can guarantee that the positions of the input vectors start from 0 and end at $L-1$ without reordering.

Then, an alignment matrix $\mathbf{A} \in \mathbb{R}^{L \times T}$ is computed based on the distances between p_j^* and q_i ($\mathbf{D} \in \mathbb{R}^{L \times T}$), and $\mathbf{X}$ is downsampled to $\mathbf{Y}$ based on $\mathbf{A}$ as follows:

$$d_{ij} = -\sigma^{-2}(p_j^* - q_i)^2, \qquad a_{ij} = \frac{\exp(d_{ij})}{\sum_{n=0}^{T-1} \exp(d_{in})}, \qquad \mathbf{Y} = \mathbf{A}@\mathbf{X}, \tag{4}$$

where σ is a hyperparameter that controls the sharpness of the alignment and @ denotes matrix multiplication. To sum up, $\mathbf{X}$ is downsampled to $\mathbf{Y}$ based on $\mathbf{A}$ calculated in VP, and we name this downsampling method AdaStride because it can be seen as a sub-sampling with adaptive strides for significantly small σ value.

In addition, we empirically observed that it is crucial to prevent every stride from being less than one (i.e., doing upsampling) to stabilize VP learning (Appendix E). In other words, every p_j^* should have at most one q_i where the p_j^* is the nearest position to the q_i. Therefore, we add an additional position loss to the original loss of each task to satisfy the condition:

$$\mathcal{L}_{pos} = \frac{1}{T-1} \sum_{t=0}^{T-2} \mathbf{max}((p_{t+1}^* - p_t^*) - 1.0,\ 0.0). \tag{5}$$

All the calculation steps of AdaStride are summarized in algorithm 1 in Appendix A.

3.2 AdaStride-F

To reduce the memory and computational cost of AdaStride, we also propose a variant of AdaStride called AdaStride-F that replaces the matrix multiplication based downsampling with a `scatter_add` operation based downsampling. In AdaStride-F, distances are calculated only from each p_j^* to its

nearest index q_i, where the q_i is equivalent to the index obtained by rounding the p_j^*:

$$d_{ij} = \begin{cases} -\sigma^{-2}(p_j^* - q_i)^2, & (q_i = \lceil p_j^* \rceil) \\ -\infty, & (q_i \neq \lceil p_j^* \rceil)) \end{cases}. \tag{6}$$

Then, the alignment matrix $\mathbf{A}$ is calculated as (4), but it is not used for matrix multiplication. Instead, since the alignment matrix $\mathbf{A}$ has only one non-zero element in a column, the matrix multiplication in (2) can be calculated using a weighting vector $\mathbf{w} = \{w_0, ..., w_{T-1}\} \in \mathbb{R}^T$ as follows:

$$w_j = \sum_{i=0}^{L-1} a_{ij}, \qquad \mathbf{y}_i = \sum_{n \in C_i} w_n \cdot \mathbf{x}_n, \qquad C_i = \{\, t \in [0, ..., T-1] \mid [p_t^*] = q_i \,\}, \tag{7}$$

where C_i is a set of indices of p_t^*s of which the nearest index is q_i, and the weighted sum is implemented using the scatter_add operation. All the calculation steps of AdaStride-F are summarized in algorithm 2 in Appendix A. In Appendix B, we provide a comparison of memory and time consumption on various downsampling methods.

4 EXPERIMENTS

In this section, we describe the experimental results on the superiority of AdaStride as a downsampling method, where we conduct three different tasks to verify its wide applicability: audio classification, automatic speech recognition (ASR), and discrete representation learning based on VQ-VAE. In all of the tasks, we use audio representaions by converting waveforms to melspectrogram features and the conversion parameters are summarized in Table 8 in Appendix D.

4.1 AUDIO CLASSIFICATION

Experimental setup We used four audio classification datasets: IEMOCAP dataset (Busso et al., 2008) for speech emotion recognition, Speech commands dataset (Warden, 2018) for speech vocabulary recognition, BirdSong dataset (Stowell et al., 2018) for birdsong detection, and TUT urban dataset (Heittola et al., 2018) for acoustic scene classification. For all of the tasks, we used the same architecture consisting of 6-layer ResBlocks (He et al., 2016b), and downsamplings were performed in the first, third, and fifth ResBlocks where the downsampling ratios were 2. At the end of the blocks, hidden representations were globally average-pooled to obtain a single vector, and it was fed to a fully-connected classifier. For more details, please refer Appendix D.

Results As shown in Table 1, our AdaStride and AdaStride-F outperformed the other downsampling methods in three of four datasets without hyperparameter tuning. However, in TUT urban dataset, spectral downsampling methods outperformed our AdaStride. We attribute this to the characteristics of the TUT urban dataset, of which audios mostly consist of plain background sound recorded at a certain place: (1) the audios have relatively uniform information density (Figure 3-(c)), so it is difficult for AdaStride to exploit many benefits from learning adaptive strides; (2) the noisy nature of the TUT urban dataset put the spectral downsampling methods in a better position, where the high-frequency noise is likely to incur aliasing in downsampling.

Computation overhead To minimize the computation overhead of the AdaStride compared to the strided convolution, we also trained a smaller model called AdaStride-S, where the AdaStride layer was only used at the most upper downsampling layer and strided convolution layers were used for the rest dowmsampling layers. In this case, it still outperformed the strided convolution in all datasets and learned effective alignments at the last AdaStride layer (Figure 11 in Appendix F), even though the time and memory overhead was only about 10% (Appendix B). Especially, it significantly improved the classification performance in IEMOCAP dataset, which shows there are tasks that benefit specifically much when using AdaStride layers.

Learned alignments Figure 3 shows that AdaStride learns effective alignments for classifying audio samples. For example, in Figures 3-(a), (b), AdaStride applies much more downsampling ratios for the silence part and less downsampling ratio for the speech part. By doing so, AdaStride can preserve much more information from the speech part, which contains most of the task-relevant

Table 1: Test accuracy results (% mean $\pm$ std over 5 runs) measured on four audio classification tasks.

Method	Dataset			
	IEMOCAP	Speech commands	TUT urban	BirdSong
Strided Conv.	46.7 $\pm$ 1.9	95.2 $\pm$ 0.1	94.5 $\pm$ 0.3	82.3 $\pm$ 0.1
Spectral Pool.	51.4 $\pm$ 1.8	95.2 $\pm$ 0.2	**95.4** $\pm$ **0.4**	83.1 $\pm$ 0.3
DiffStride	51.1 $\pm$ 1.3	95.4 $\pm$ 0.1	**95.4** $\pm$ **0.3**	82.8 $\pm$ 0.2
AdaStride	**52.2** $\pm$ **1.0**	**95.9** $\pm$ **0.1**	95.1 $\pm$ 0.6	**83.3** $\pm$ **0.5**
AdaStride-F	51.9 $\pm$ 1.2	**95.9** $\pm$ **0.1**	95.1 $\pm$ 0.5	**83.3** $\pm$ **0.4**
AdaStride-S	51.0 $\pm$ 1.9	95.4 $\pm$ 0.1	94.6 $\pm$ 0.5	82.9 $\pm$ 0.4

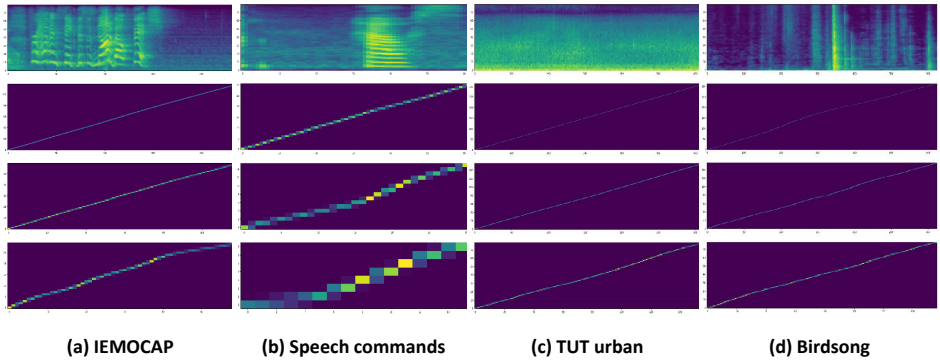


Figure 3: Melspectrograms and alignments obtained from the AdaStride layers when classifying the melspectrograms. Each column represents each dataset.

information. In contrast, the alignments in Figure 3-(c) have the shape of an almost straight diagonal line, and it means that AdaStride applies almost the same downsampling ratio to the entire input. Therefore, there is less gain from using AdaStride for the dataset such as the TUT urban dataset.

Relationship between σ and aliasing In equations (4) and (6), the hyperparameter σ controls the sharpness of the alignment, where the smaller the σ is, the sharper the alignment matrix is. In other words, we can balance the trade-off between preserving high-frequency components and anti-aliasing by adjusting the σ value. Therefore, we conducted an experiment in which we trained various AdaStride models with different σ values on the audio classification tasks. From the experiment, we found out that it was crucial to select a proper σ value depending on which dataset was used. For example, AdaStride with $\sigma^{-2} = 0.1$ showed the best performance on the TUT urban dataset and even outperformed the spectral downsampling methods, unlike the results in Table 1. Meanwhile, AdaStride with $\sigma^{-2} = 5.0$ achieved the best performance on the IEMOCAP and Birdsong datasets, which means that preserving high-frequency components is preferred to anti-aliasing on those datasets compared to the TUT urban dataset. To see the shape of the alignments learned for different σ values, please refer Figure 12 in Appendix F.

4.2 Automatic speech recognition

Experimental setup In this experiment, we compare various downsampling methods in ASR by using them in a subsampling module to reduce the time resolution of an input. Specifically, the subsampling module consists of two (Conv-Pool-SiLu) blocks where each block downsamples the input by 2. Then, the output of the subsampling module is fed to an encoder consisting of the 16 Squeezeformer (Kim et al., 2022) blocks. Here, we excluded DiffStride because its overall downsampling ratio keeps varying during training. We trained ASR models with different downsampling methods based on CTC loss (Graves et al., 2006) on LibriSpeech ASR corpus (Panayotov et al., 2015), and we compared phoneme error rate (PER) and word error rate (WER) of the models. We used a 1k

Table 2: Test accuracy results (% mean $\pm$ std over 5 runs) measured on four audio classification tasks.

	Dataset			
	IEMOCAP	Speech commands	TUT urban	Birdsong
$\sigma^{-2} = 0.1$	50.4 ± 1.7	93.0 ± 0.1	$\mathbf{95.6 \pm 0.3}$	76.1 ± 0.6
$\sigma^{-2} = 1.0$	52.1 ± 0.7	95.6 ± 0.1	95.2 ± 0.5	82.3 ± 0.7
$\sigma^{-2} = 5.0$	$\mathbf{52.2 \pm 1.0}$	95.9 ± 0.1	95.1 ± 0.6	$\mathbf{83.3 \pm 0.5}$
$\sigma^{-2} = 10.0$	52.2 ± 1.0	$\mathbf{96.0 \pm 0.1}$	95.2 ± 0.6	83.3 ± 0.2

subword tokenizer trained using WordPiece algorithm (Wu et al., 2016) on the train datasets. Also, we used SpecAugment (Park et al., 2019) with one frequency-mask with mask parameter ($F = 27$) and five time-masks with maximum time-mask ratio ($p_S = 0.05$). For more details such as audio representation and training parameters, please refer Appendix D.

Table 3: PER and WER results (%) measured on the LibriSpeech dataset.

	dev-clean		dev-other		test-clean		test-other	
Method	PER	WER	PER	WER	PER	WER	PER	WER
Strided Conv.	1.46	4.87	4.66	12.29	1.55	4.88	4.43	11.91
Spectral Pool.	1.53	4.54	4.79	11.63	1.59	4.62	4.62	11.53
AdaStride	1.40	**4.05**	**4.56**	**10.98**	**1.47**	**4.11**	4.28	**10.75**
AdaStride-F	**1.38**	4.50	4.62	11.71	1.53	4.38	**4.26**	11.50

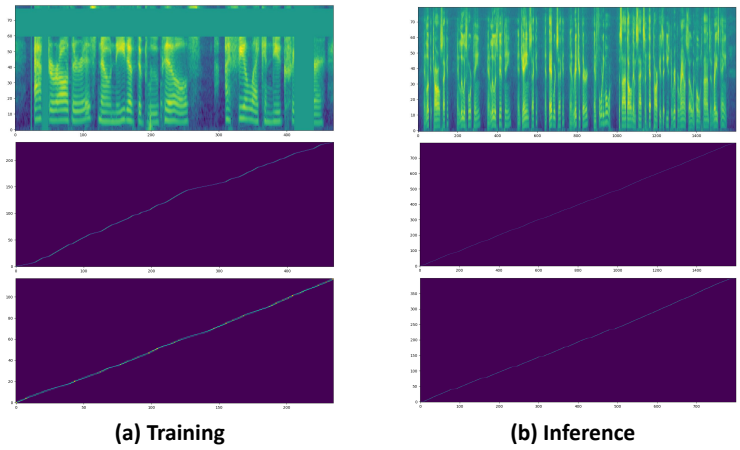


Figure 4: Melspectrograms and alignments obtained from the AdaStride layers of the ASR networks. During training, SpecAugment is applied to the input features.

Results As shown in Table 3, our AdaStride and AdaStride-F outperformed the other downsampling methods for both PER and WER for all data subsets while combined with the state-of-the-art ASR Squeezeformer architecture. Since recognizing a speech is to extract a word seqeunce (sequential data) from an audio input (time series data), applying varying downsampling ratios to the audios in ASR is still effective rather than hurting the performance. In addition, the alignments that AdaStride learns show that, even if SpecAugment is used during training, it learns effective adaptive strides. For instance, it is shown in Figure 4 that it compresses the silence parts and the time-masked parts of input much more than the other parts containing phonemes.

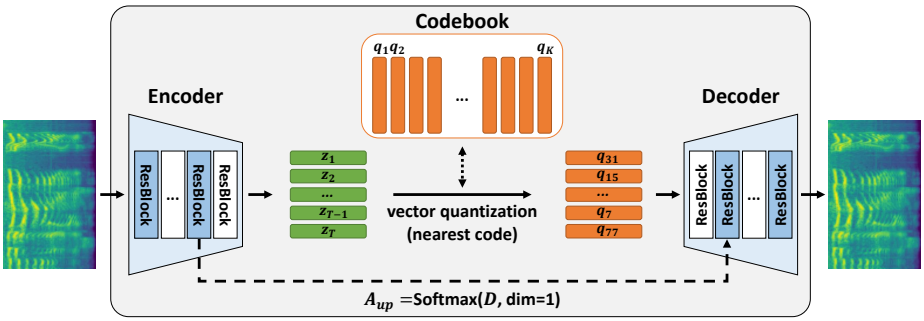


Figure 5: VQ-VAE architecture. (Down- / Up-) samplings are conducted in the colored ResBlocks.

4.3 Discrete representation learning

In this section, we demonstrate an experiment conducted based on VQ-VAE (Van Den Oord et al., 2017) to see whether AdaStride is superior in discrete representation learning over the other downsampling methods. Here, we compare AdaStride with strided convolution and spectral pooling, all of which use a fixed overall downsampling ratio. VQ-VAE is one of the representation learning frameworks that learns to represent data with discrete codes as shown in Figure 5.

Experimental setup In this experiment, we used LJSpeech-1.1 dataset (Ito & Johnson, 2017), and we adopted 6-ResBlock Encoder-Decoder architecture as our baseline model, where downsamplings are conducted alternately in the encoder ResBlocks, and nearest neighbor upsamplings are conducted alternately in the decoder ResBlocks. As a result, it learns to obtain code representations of which time steps are reduced by 1/8 from the original time steps of a melspectrogram. When we used AdaStride, we used it only for the downsampling conducted in the second to last encoder ResBlock for simplicity. This is because AdaStride encodes parts of information using the alignment matrix so the alignment information should be passed to the decoder as well as the discrete code representations. Specifically, we passed an alignment matrix $\mathbf{A_{up}}$ to the second decoder ResBlock for upsampling where $\mathbf{A_{up}}$ was calculated by applying softmax to the distance matrix $\mathbf{D}$ along the first dimension. Therefore, AdaStride-F was excluded in this experiment because applying softmax operation to $\mathbf{D}$ along the first dimension is meaningless for AdaStride-F. For more details such as audio representation, training parameters, architecture, and training of VQ-VAE, please refer Appendix C and D.

Table 4: PER-R and PER-C results (%) measured on the LJSpeech-1.1 dataset.

Method	PER-R	PER-C
Strided Conv.	11.44	21.23
Spectral Pool.	18.14	25.53
AdaStride	**10.85**	**19.51*** / 26.76**

*, **: each of these represents whether it uses alignments or not

Results To see the effectiveness of AdaStride in discrete representation learning, we evaluated the representations that the VQ-VAE models using different downsampling methods learn. First, we evaluated the performance of VQ-VAE itself by comparing the reconstruction performance based on phoneme error rate (PER-R). For this, we generated a hundred audio samples by applying HiFi-GAN (Kong et al., 2020) to the melspectrograms reconstructed by each VQ-VAE model. Then, we measured PER-R on the generated audio samples via the Google Cloud Speech-to-Text API.[1] Second, we evaluated the quality of the discrete representations by training ASR models using the learned discrete representations and measuring phoneme error rate (PER-C). In the case of AdaStride, we separated the cases of whether the alignments that AdaStride learns are used or not used in ASR training. Specifically, the representations are upsampled by 2 based on the learned alignments or by

[1] `https://cloud.google.com/speech-to-text`

repeating each representation twice as shown in Figure 9 in Appendix D. The result of the experiment is shown in Table 4, where our AdaStride outperforms the strided convolution and spectral pooling in terms of both PER-R and PER-C. However, since the code representations and alignments are not learned targeting ASR, PER-C becomes worse when the alignments are not used in ASR training. To see the alignments that AdaStride learns, please refer Figure 13 in Appendix F.

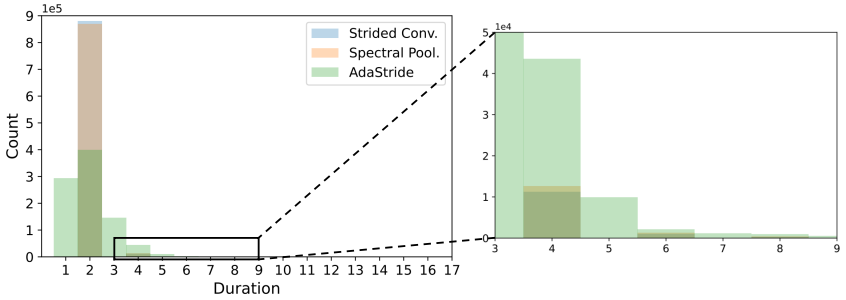


Figure 6: The counts of durations of codes that different VQ-VAE models learn.

Table 5: The number of unique codes used in encoding the LJSpeech-1.1 dataset.

	Strided Conv.	Spectral Pool.	AdaStride
Unique codes	390 / 512	383 / 512	365 / 512

Encoding data with codes and durations To see how AdaStride is effective compared to the other downsampling methods, we analyzed how the data encoding is learned in terms of codes and their durations. Here, durations of codes mean lengths of consecutive groups of equivalent codes when codes are upsampled back to the length right before the last downsampling. For AdaStride, the code durations were calculated after upsampling the codes using the alignment matrix which was converted to a one-hot matrix via argmax operation. Figure 6 illustrates the counts of durations of codes that the different downsampling methods learn. As is expected, AdaStride learned to represent data with codes with variable durations so it can encode information-rich parts of data using more codes in finer resolution. In contrast, strided convolution and spectral pooling ineffectively encoded data with codes most of which have durations of two. Furthermore, the fact that there were no durations longer than two represents that even a long-lasting section in a data sample was encoded using various codes. Therefore, this way of encoding can lead to a waste of codebook capacity, and it can result in the performance drop when the size of the codebook decreases. Indeed, Table 5 illustrates how many unique codes existed when each VQ-VAE model encoded the full LJSpeech-1.1 dataset, where AdaStride used the least number of codes.

5 Conclusion

In this paper, we proposed a novel downsampling method called AdaStride, which learns to apply adaptively varying downsampling ratios within a sequential data sample via the vector positioning process. In this way, it can preserve information for task-relevant parts of a data sample better than the fixed overall downsampling rate. In audio classification, AdaStride outperformed other standard downsampling methods on three of four datasets without hyperparameter tuning and on all of the datasets with hyperparameter tuning. In addition, even when we minimized the memory increase of AdaStride compared to strided convolution, it still achieved significant performance improvement, especially on the IEMOCAP dataset. In ASR, AdaStride also outperformed the other downsampling methods in terms of both WER and PER for all data subsets. In VQ-VAE, we saw that AdaStride is superior in reconstructing the input melspectrogram due to its effective utilization of codes and durations, and it indeed leads to better representations for sub-tasks such as ASR.

REPRODUCIBILITY

To help readers reproduce the experiments, we provided a separate paragraph describing the experimental setup for each sub-task. Also, we elaborated on other details such as data preprocessing, training parameters, and the architectural details in appendices with many figures describing the architectures for precise re-implementation. Furthermore, we provided a source code of AdaStride as supplementary material, where you can reproduce the audio classification experiment using the AdaStride. Since we took great care to modularize the AdaStride layer while writing the code, you can also easily employ the AdaStride in other tasks.

REFERENCES

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1409.0473.

Carlos Busso, Murtaza Bulut, Chi-Chun Lee, Abe Kazemzadeh, Emily Mower, Samuel Kim, Jeannette N Chang, Sungbok Lee, and Shrikanth S Narayanan. Iemocap: Interactive emotional dyadic motion capture database. *Language resources and evaluation*, 42(4):335–359, 2008.

William Chan, Navdeep Jaitly, Quoc V. Le, and Oriol Vinyals. Listen, attend and spell. *CoRR*, abs/1508.01211, 2015. URL http://arxiv.org/abs/1508.01211.

Hyeong-Seok Choi, Juheon Lee, Wansoo Kim, Jie Lee, Hoon Heo, and Kyogu Lee. Neural analysis and synthesis: Reconstructing speech from self-supervised representations. *Advances in Neural Information Processing Systems*, 34, 2021.

Ronan Collobert, Awni Hannun, and Gabriel Synnaeve. Word-level speech recognition with a letter to word encoder. In *International Conference on Machine Learning*, pp. 2100–2110. PMLR, 2020.

Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, and Ilya Sutskever. Jukebox: A generative model for music. *arXiv preprint arXiv:2005.00341*, 2020.

Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12873–12883, 2021.

Kunihiko Fukushima and Sei Miyake. Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition. In *Competition and cooperation in neural nets*, pp. 267–285. Springer, 1982.

Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pp. 369–376, 2006.

Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented Transformer for Speech Recognition. In *Proc. Interspeech 2020*, pp. 5036–5040, 2020. doi: 10.21437/Interspeech.2020-3015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pp. 630–645. Springer, 2016b.

Toni Kristian Heittola, Annamaria Mesaros, and Tuomas Virtanen, 2018. Heittola, Toni, Mesaros, Annamaria, & Virtanen, Tuomas. (2018). TUT Urban Acoustic Scenes 2018, Development dataset [Data set]. Zenodo. https://doi.org/10.5281/zenodo.1228142.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997.

Keith Ito and Linda Johnson. The lj speech dataset. https://keithito.com/LJ-Speech-Dataset/, 2017.

Shigeki Karita, Nanxin Chen, Tomoki Hayashi, Takaaki Hori, Hirofumi Inaguma, Ziyan Jiang, Masao Someki, Nelson Enrique Yalta Soplin, Ryuichi Yamamoto, Xiaofei Wang, et al. A comparative study on transformer vs rnn in speech applications. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pp. 449–456. IEEE, 2019.

Sehoon Kim, Amir Gholami, Albert Shaw, Nicholas Lee, Karttikeya Mangalam, Jitendra Malik, Michael W Mahoney, and Kurt Keutzer. Squeezeformer: An efficient transformer for automatic speech recognition. *arXiv preprint arXiv:2206.00888*, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR (Poster)*, 2015. URL http://arxiv.org/abs/1412.6980.

Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. *Advances in Neural Information Processing Systems*, 33:17022–17033, 2020.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Yann LeCun, Bernhard Boser, John Denker, Donnie Henderson, Richard Howard, Wayne Hubbard, and Lawrence Jackel. Handwritten digit recognition with a back-propagation network. *Advances in neural information processing systems*, 2, 1989.

Junjie Li and Ding Liu. Information bottleneck theory on convolutional neural networks. *arXiv preprint arXiv:1911.03722*, 2019.

Yuanchao Li, Tianyu Zhao, and Tatsuya Kawahara. Improved End-to-End Speech Emotion Recognition Using Self Attention Mechanism and Multitask Learning. In *Proc. Interspeech 2019*, pp. 2803–2807, 2019. doi: 10.21437/Interspeech.2019-2594.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=Bkg6RiCqY7.

Youxuan Ma, Zongze Ren, and Shugong Xu. RW-Resnet: A Novel Speech Anti-Spoofing Model Using Raw Waveform. In *Proc. Interspeech 2021*, pp. 4144–4148, 2021. doi: 10.21437/Interspeech.2021-438.

Xudong Mao, Qing Li, Haoran Xie, Raymond YK Lau, Zhen Wang, and Stephen Paul Smolley. Least squares generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2794–2802, 2017.

Chenfeng Miao, Liang Shuang, Zhengchen Liu, Chen Minchuan, Jun Ma, Shaojun Wang, and Jing Xiao. Efficienttts: An efficient and high-quality text-to-speech architecture. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7700–7709. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/miao21a.html.

Takeru Miyato, Toshiki Kataoka, Masanori Koyama, and Yuichi Yoshida. Spectral normalization for generative adversarial networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=B1QRgziT-.

H. Nyquist. Certain topics in telegraph transmission theory. *Transactions of the American Institute of Electrical Engineers*, 47(2):617–644, 1928. doi: 10.1109/T-AIEE.1928.5055024.

Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pp. 5206–5210. IEEE, 2015.

Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D. Cubuk, and Quoc V. Le. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. In *Proc. Interspeech 2019*, pp. 2613–2617, 2019. doi: 10.21437/Interspeech.2019-2680.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1511.06434.

Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.

Rachid Riad, Olivier Teboul, David Grangier, and Neil Zeghidour. Learning strides in convolutional neural networks. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=M752z9FKJP.

Oren Rippel, Jasper Snoek, and Ryan P Adams. Spectral representations for convolutional neural networks. In C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/536a76f94cf7535158f66cfbd4b113b6-Paper.pdf.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

Tim Salimans and Durk P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *Advances in neural information processing systems*, 29, 2016.

D. Stowell, Y. Stylianou, M. Wood, H. Pamuła, and H. Glotin. Automatic acoustic detection of birds through deep learning: the first bird audio detection challenge. *Methods in Ecology and Evolution*, 2018. URL https://arxiv.org/abs/1807.05812].

Gabriel Synnaeve, Qiantong Xu, Jacob Kahn, Tatiana Likhomanenko, Edouard Grave, Vineel Pratap, Anuroop Sriram, Vitaliy Liptchinsky, and Ronan Collobert. End-to-end asr: from supervised to semi-supervised learning with modern architectures. *arXiv preprint arXiv:1911.08460*, 2019.

Aaron Van Den Oord, Oriol Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

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

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Seunghyun Yoon, Seokhyun Byun, Subhadeep Dey, and Kyomin Jung. Speech emotion recognition using multi-hop attention mechanism. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 2822–2826. IEEE, 2019.

Richard Zhang. Making convolutional networks shift-invariant again. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7324–7334. PMLR, 2019. URL http://proceedings.mlr.press/v97/zhang19a.html.

Ya-Jie Zhang, Shifeng Pan, Lei He, and Zhen-Hua Ling. Learning latent representations for style control and transfer in end-to-end speech synthesis. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6945–6949. IEEE, 2019.

A ALGORITHMS

Algorithm 1: Pseudo-code describing how AdaStride works

Data:
$\mathbf{X} \in \mathbb{R}^{T \times D}$: input of AdaStride, a sequential data consisting of T time steps
s: overall stride factor that determines the ratio of the length of the input to the output
σ: hyperparameter that controls the sharpness of the alignment matrix
Result:
$\mathbf{Y} \in \mathbb{R}^{L \times D}$: output of AdaStride, a sequential data consisting of $L = \lceil T/s \rceil$ time steps
$\mathcal{L}_{pos}$: position loss that stabilizes the training of AdaStride

```
/* Caculate normalized positions of the input time steps      */
```
$\Delta\mathbf{p} \leftarrow$ `Exponential(PositionalFC(`$\mathbf{X}$`))`;
$\mathbf{p} \leftarrow$ `Cumsum(`$\Delta\mathbf{p}$, $\mathsf{dim} = 0$`)`;
$\mathbf{p}^* \leftarrow (\mathbf{p} - \mathbf{p}[0:1])/(\mathbf{p}[T-1:] - \mathbf{p}[0:1]) * (L-1)$;

```
/* Construct an alignment matrix based on p* and q            */
```
$\mathbf{q} \leftarrow \mathsf{range}(0, L)$;
$\mathbf{D} \leftarrow -\sigma^{-2}(\mathbf{p}^* - \mathbf{q})^2.\mathsf{transpose}()$;
$\mathbf{A} \leftarrow$ `Softmax(`$\mathbf{D}$, $\mathsf{dim} = 1$`)`;

```
/* Perform downsampling and calculate the position loss       */
```
$\mathbf{Y} \leftarrow \mathbf{A}@\mathbf{X}$;
$\mathcal{L}_{pos} \leftarrow \frac{1}{T-1} \sum_{i=0}^{T-2} \mathbf{max}((\mathbf{p}^*[i+1] - \mathbf{p}^*[i]) - 1.0,\ 0.0)$

Algorithm 2: Pseudo-code describing how AdaStride-F works

Data:
$\mathbf{X} \in \mathbb{R}^{T \times D}$: input of AdaStride, a sequential data consisting of T time steps
s: overall stride factor that determines the ratio of the length of the input to the output
σ: hyperparameter that controls the sharpness of the alignment matrix
Result:
$\mathbf{Y} \in \mathbb{R}^{L \times D}$: output of AdaStride, a sequential data consisting of $L = \lceil T/s \rceil$ time steps
$\mathcal{L}_{pos}$: position loss that stabilizes the training of AdaStride

```
/* Caculate normalized positions of the input time steps      */
```
$\Delta\mathbf{p} \leftarrow$ `Exponential(PositionalFC(`$\mathbf{X}$`))`;
$\mathbf{p} \leftarrow$ `Cumsum(`$\Delta\mathbf{p}$, $\mathsf{dim} = 0$`)`;
$\mathbf{p}^* \leftarrow (\mathbf{p} - \mathbf{p}[0:1])/(\mathbf{p}[T-1:] - \mathbf{p}[0:1]) * (L-1)$;

```
/* Construct an alignment matrix based on p*                  */
```
$\mathbf{d} \leftarrow -\sigma^{-2}(\mathbf{p}^* - [\mathbf{p}^*])^2$;
$\mathbf{D} \leftarrow -\infty(L, T)$;
$\mathbf{D} \leftarrow \mathbf{D}.\mathsf{scatter}(\mathsf{dim} = 0,\ \mathsf{index} = [\mathbf{p}^*].\mathsf{t}(),\ \mathsf{src} = \mathbf{d}.\mathsf{t}())$;
$\mathbf{A} \leftarrow$ `Softmax(`$\mathbf{D}$, $\mathsf{dim} = 1$`)`;
$\mathbf{W} \leftarrow \mathbf{A}.\mathsf{sum}(\mathsf{dim} = 0).\mathsf{unsqueeze}(-1)$;

```
/* Perform downsampling and calculate the position loss       */
```
$\mathbf{Y} \leftarrow 0(L, D)$;
$\mathbf{Y} \leftarrow \mathbf{Y}.\mathsf{scatter_add}(\mathsf{dim} = 0,\ \mathsf{index} = [\mathbf{p}^*].\mathsf{repeat}(1, \mathsf{D}),\ \mathsf{src} = \mathbf{W} \cdot \mathbf{X})$;
$\mathcal{L}_{pos} \leftarrow \frac{1}{T-1} \sum_{i=0}^{T-2} \mathbf{max}((\mathbf{p}^*[i+1] - \mathbf{p}^*[i]) - 1.0,\ 0.0)$

B MEMORY AND TIME CONSUMPTION

Table 6: Memory and time consumption per epoch for different downsampling methods relative to strided convolution.

		Strided Conv.	Spectral Pool.	AdaStride	AdaStride-F	AdaStride-S
Training	Mem	1.00	1.38	2.53	2.34	1.11
	Time	1.00	1.25	1.38	1.20	1.08
Inference	Mem	1.00	1.00	1.24	1.25	1.08
	Time	1.00	1.70	1.66	1.65	1.08

We measured the memory and time consumption of the different downsampling methods on the IEMOCAP dataset. Specifically, Table 6 shows the memory and time consumption used to process an epoch for training and inference for different downsampling methods relative to strided convolution.

C VQ-VAE

VQ-VAE In this paper, VQ-VAE(Van Den Oord et al., 2017; Razavi et al., 2019) is used to represent a melspectrogram with a sequence of positive integers called codes. During training, the encoder obtains a sequence of continuous vector representations ($z_e(\mathbf{X})$) from a melspectrogram. Then, each of the vector representations is replaced with one among the set of code representations (codebook) based on the nearest neighbor lookup. Lastly, the quantized vector representations ($z_q(\mathbf{X})$) are passed to the decoder, and the decoder learns to reconstruct the original input based on mean absolute error ($\mathcal{L}_{mae}$).

To successfully achieve the discrete representation learning, several techniques are used in VQ-VAE training: (1) during backpropagation, the gradients from the decoder input $z_q(\mathbf{X})$ are passed to the encoder output $z_e(\mathbf{X})$, so encoder can learn to obtain better representations; (2) there are two additional losses used to minimize the gap arisen from the vector quantization process:

$$\mathcal{L}_{codebook}(\mathbf{X}) = \frac{1}{S}\sum_{s} ||z_q(\mathbf{X})_s - sg[z_e(\mathbf{X})_s]||_2^2, \tag{8}$$

$$\mathcal{L}_{commit}(\mathbf{X}) = \frac{1}{S}\sum_{s} ||sg[z_q(\mathbf{X})_s] - z_e(\mathbf{X})_s||_2^2, \tag{9}$$

where S is the length of the representations and sg denotes the stop-gradient operation that passes zero gradients during backpropagation.

Furthermore, to prevent the codes from being not used and to speed up training, we adopted several more techniques called code restart and exponential moving average (EMA) code update (Dhariwal et al., 2020). Code restart is a method of replacing a code that is rarely used during training with a representation randomly sampled from $z_e(\mathbf{X})$. EMA code update is to update code representations based on EMA instead of the codebook loss $\mathcal{L}_{codebook}$ to speed up training. As a result, the total loss of VQ-VAE becomes as follows:

$$\mathcal{L}_{vqvae}(\mathbf{X}) = \mathcal{L}_{mae}(\mathbf{X}, G(\mathbf{X})) + \alpha * \mathcal{L}_{commit}(\mathbf{X}), \tag{10}$$

where G is the VQ-VAE model and $G(\mathbf{X})$ is the melspectrogram reconstructed by the model.

Adversarial training To guide the code representations to capture more perceptually important information, we employed an additional discriminator and adopted the GAN approach in our VQ-VAE training (Esser et al., 2021). We adopted the architecture of the discriminator following (Choi et al., 2021) and it is shown in Figure 7. For GAN loss, we used LSGAN losses (Mao et al., 2017) with

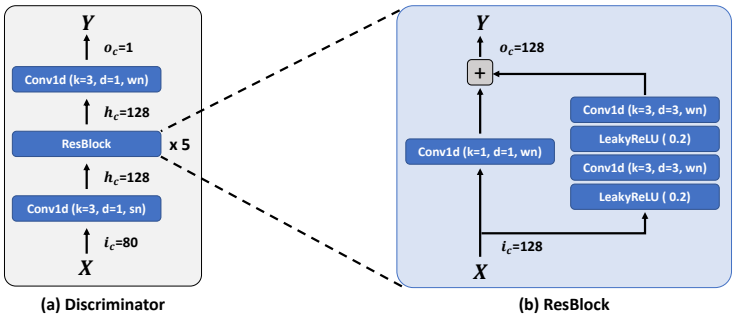


Figure 7: Discriminator architecture. **'sn'** denotes spectral normalization (Miyato et al., 2018) and **'wn'** denotes weight normalization (Salimans & Kingma, 2016).

additional feature-matching loss, and they are as follows:

$$\mathcal{L}_{adv}(D;G)(\mathbf{X}) = \mathbb{E}_{\mathbf{X}}\left[(D(\mathbf{X})-1)^2 + (D(G(\mathbf{X})))^2\right], \tag{11}$$

$$\mathcal{L}_{adv}(G;D)(\mathbf{X}) = \mathbb{E}_{\mathbf{X}}\left[(D(G(\mathbf{X}))-1)^2\right], \tag{12}$$

$$\mathcal{L}_{fm}(G;D)(\mathbf{X}) = \mathbb{E}_{\mathbf{X}}\left[\sum_{l=1}^{5}\frac{1}{N_l}||D^l(\mathbf{X}) - D^l(G(\mathbf{X}))||_1\right], \tag{13}$$

$$\tag{14}$$

where l denotes l-th layer of the discriminator, and D_l and N_l denote the features and the number of features in the l-th layer of the discriminator, respectively. As a result, the final loss of VQ-VAE becomes as follows:

$$\mathcal{L}_D = \mathcal{L}_{adv}(D;G), \tag{15}$$

$$\mathcal{L}_G = \mathcal{L}_{vqvae} + \beta * \mathcal{L}_{adv}(G;D) + \gamma * \mathcal{L}_{fm}(G;D). \tag{16}$$

D ADDITIONAL DETAILS

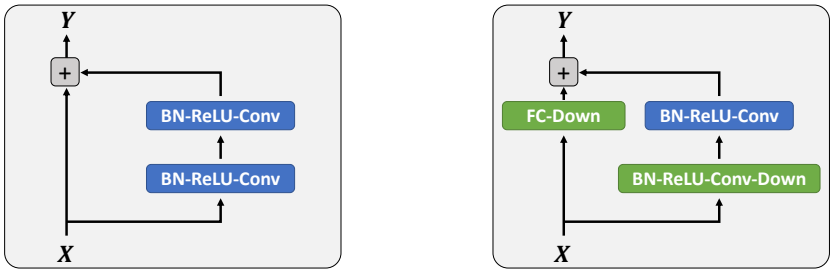
Dataset In Table 7, the information about the datasets used in this paper is demonstrated.

Table 7: Information about the datasets.

	License	Personal information	Offensive content
IEMOCAP	Custom (non-commercial)*	✗	✗
Speech commands	CC-BY	✗	✗
TUT urban	Custom (non-commercial)**	✗	✗
Birdsong	CC-BY 4.0	✗	✗
LibriSpeech	CC-BY 4.0	✗	✗
LJSpeech-1.1	CC0	✗	✗

*: `https://sail.usc.edu/iemocap/Data_Release_Form_IEMOCAP.pdf`
**: `https://zenodo.org/record/1237793/files/LICENSE?download=1`

Architecture of ResBlock We adopted PreAct-ResNet block (He et al., 2016b) shown in Figure 8 for audio classification and discrete representation learning. Specifically, we compare various downsampling methods by replacing the strided convolution of the ResNet block with a normal convolution followed by one of the downsampling layers, as shown in Figure 8. When the downsampling layer is AdaStride, the vector positioning process is conducted in the main path of ResNet, and in the shorcut path of ResNet, downsampling is conducted using the alignment matrix obtained in the main path.



(a) ResBlock without downsampling

(b) ResBlock with downsampling

Figure 8: ResBlock architectures with and without downsampling.

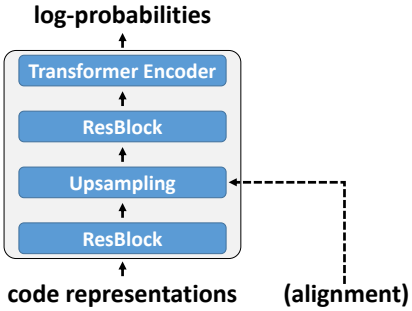


Figure 9: ASR network architecture trained with code representations.

ASR network architecture trained with code representations We trained ASR networks (Figure 9) with code sequences obtained via VQ-VAE encoder based on CTC loss. Also, we optionally used alignments that were obtained by AdaStride Encoder, and if alignments were not used, we expanded the code sequences by two based on nearest neighbor upsampling to match the time resolution of the code sequences equally for a fair experiment. Each network was trained for 100k iterations with a batch size of 32 and a learning rate of 1e-3 using AdamW optimizer (Loshchilov & Hutter, 2019).

Training instability and layer normalization In ASR and VQ-VAE training, we sometimes found out that it was unstable to train models using AdaStride due to the exponential function existing in the vector positioning process. However, we could easily resolve this problem by putting a layer normalization layer (Ba et al., 2016) before the AdaStride layer.

Hyperparameters In Table 8, the hyperparameters used in each task are demonstrated.

Table 8: Hyperparameters used in each task.

	Audio classification	Speech recognition	VQ-VAE
hidden dimension	256	144	512
kernel size	3	3	3
σ^{-2}	5.0	5.0	10.0
K (# of codes)	-	-	512
α	-	-	0.1
β	-	-	1.0
γ	-	-	1.0
sampling rate	16000	16000	22050
filter banks	80	80	80
windows	1024	400	1024
hops	256	160	256
optimizer	Adam*	AdamW**	Adam
learning rate	1e-4	5e-4	3e-4
batch size	128	256	64

*: Kingma & Ba (2015)
**: Loshchilov & Hutter (2019)

E POSITION LOSS

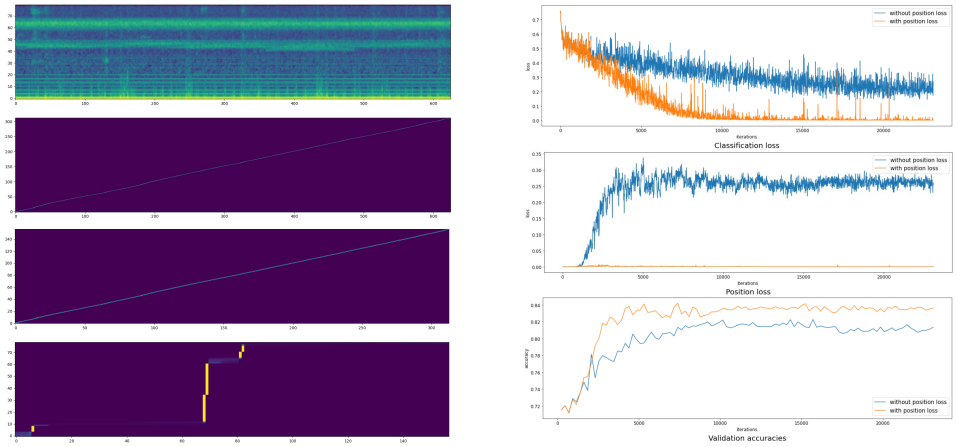


Figure 10: Alignments and the training losses and validation accuracy curves when position loss is used or not.

We empirically observed that, witout position loss, training becomes unstable leading to radical alignment learning and poor performance. For instance, Figure 10 shows the training logs when a ResNet equipped with AdaStride layers is trained to solve the birdsong task: (1) the alignments that AdaStride layers learn; (2) training losses and validation accuracy curves when position loss is used or not.

F ALIGNMENT SAMPLES

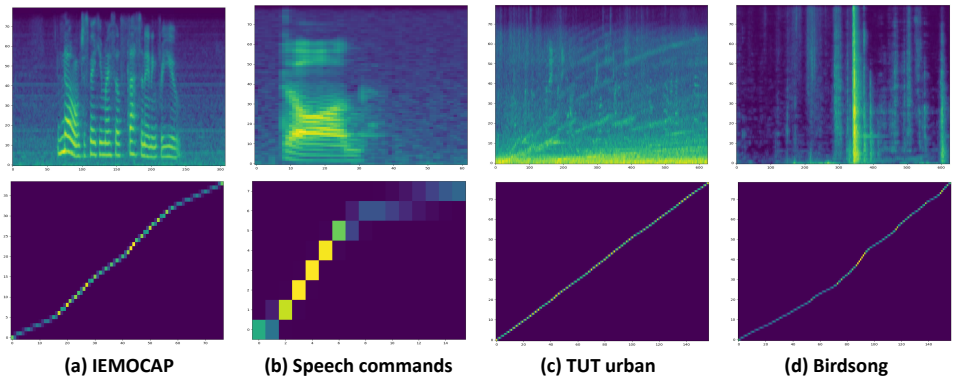


Figure 11: Melspectrograms and alignments obtained from the AdaStride layers of the audio classification models when the AdaStride layers are used only for the last downsampling layers.

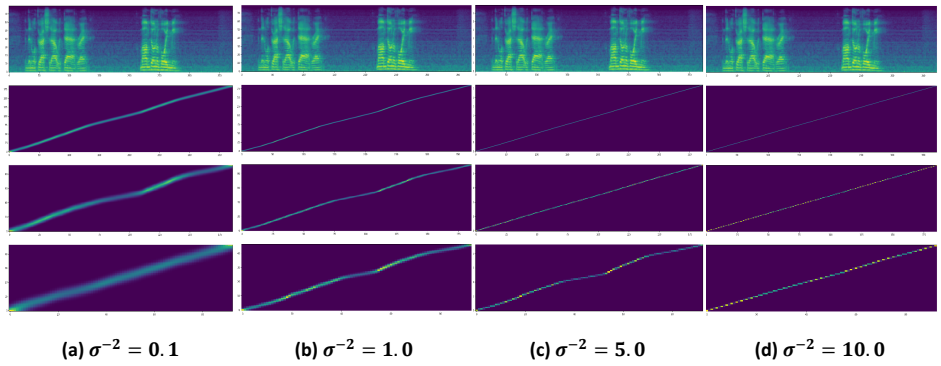


Figure 12: Melspectrograms and alignments obtained from the AdaStride layer of the models with varying sigma values, which are trained on IEMOCAP dataset.

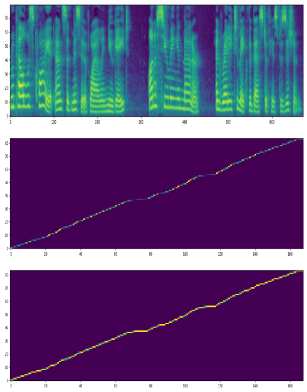


Figure 13: Melspectrograms and alignments obtained from the ResBlocks of the VQ-VAE. The alignment at the bottom is the alignment used for upsampling and it is transposed here.