Anti-aliasing regularization in stacking layers

Antoine Bruguier, Ananya Misra, Arun Narayanan, Rohit Prabhavalkar

Google LLC, USA

{tonybruguier,amisra,arunnt,prabhavalkar}@google.com

Abstract

Shift-invariance is a desirable property of many machine learning models. It means that delaying the input of a model in time should only result in delaying its prediction in time. A model that is shift-invariant, also eliminates undesirable side effects like frequency aliasing. When building sequence models, not only should the shift-invariance property be preserved when sampling input features, it must also be respected inside the model itself. Here, we study the impact of the commonly used stacking layer in LSTM-based ASR models and show that aliasing is likely to occur. Experimentally, by adding merely 7 parameters to an existing speech recognition model that has 120 million parameters, we are able to reduce the impact of aliasing. This acts as a regularizer that discards frequencies the model shouldn't be relying on for predictions. Our results show that under conditions unseen at training, we are able to reduce the relative word error rate by up to 5%.

Index Terms: aliasing, sampling theorem, stacking layers, regularization, speech recognition

1. Introduction

Frequencies that are different in the original signal can become indistinguishable after down-sampling an audio signal. This phenomenon is known as *aliasing* [1]. Aliasing also occurs when down-sampling an image and it can be avoided using 2D low-pass filters [2, 3]. Any time a signal is indexed with time-like variables and we downsample along that axis (e.g., by pooling, or sub-sampling), aliasing should be carefully considered.

An important result of signal processing is that by not taking aliasing into account, we can make a process non-shift invariant [1]. To define shift-invariance, consider the transformation F of a signal $x(t)$ into $y(t)$:

$$x(t) \xrightarrow{F} y(t)$$

The transformation is shift-invariant if for any shift τ we also have:

$$x(t-\tau) \xrightarrow{F} y(t-\tau)$$

As a simple illustration of the relationship between aliasing and shift-invariance, consider the continuous time signals $x(t) = \sin(\pi(t-\tau))$ and the process F that consists of sampling the signal for every integer value of $t = n$ and then reconstructing the signal $y(t)$ from the samples. In this case, the signal can be reconstructed perfectly by unambiguously computing τ. However, if we had instead allowed higher-frequency signals, such as $x(t) = \sin(4\pi(t-\tau))$, it would become impossible to distinguish $\tau = 0$ from $\tau = 1$ as for every integer n, $\sin(4\pi(n-0)) = \sin(4\pi(n-1))$

Failure to be shift-invariant can cause undesirable artifacts. Moiré patterns [4] are a famous example of a failure for images. In the time domain, the wheel of a car can appear to spin backward in movies shot on film [5].

Shift invariance is also a desirable property for many machine learning tasks. An image identification task should have the same prediction if the input image is shifted by one pixel. Similarly, an automatic speech recognition (ASR) task should eventually decode the same transcript if the input is shifted in time.

Aliasing is typically avoided by removing the high frequencies in the signal using a low-pass filter prior to down-sampling. The cut-off frequency to use is dictated by the Shannon-Nyquist theorem [6]. However, when building machine learning models, it is not a sufficient condition as we must make sure that the internal transformations of the signal *inside* the neural network are also shift-invariant. Some layers, such as feed forward networks are natively integer shift-invariant because they treat each time sample independently, but many are not.

Shift-invariance has been studied in the context of pooling convolutional neural networks (CNN) for acoustic models [7] and image processing [8]. Results appear to be more encouraging with image processing, potentially because they often have sharper contrasts leading to high frequency components, while high frequencies in speech do not necessarily carry as much information as low frequencies. Other studies [9] in image processing have also shown limited improvements so any improvement must be validated experimentally.

The present work focuses on stacking layers. Stacking layers, or time-reduction layers, are typically used in end-to-end ASR models to reduce the sampling frequency of features within the model. For example, in [10] the authors employ pyramidal layers and in [11, 12, 13] the authors use a stacking layer. This helps reduce overall computation, while also partly addressing the mismatch in total number of input and output tokens.

Stacking layers take a sequence of N vectors of dimension D and output a sequence of $\frac{N}{K}$ vectors of dimension $K \times D$. The parameter K is an integer, and is the rate at which the sampling frequency is reduced. We will show that this transformation results in a violation of the shift-invariance property. Subsequently, we introduce a simple low-pass filtering layer within the model to prevent aliasing when using stacking layers, and experimentally show that this helps improve performance in mismatched test conditions. In effect, this could be seen as a form of regularization where undesired parts of the signal (the high frequencies) are forced to zero.

The rest of the paper is organized as follows. In Section 2, we present an analysis that shows why stacking layers are not shift invariant. In section 3 we present the base model that we want to improve. In section 4 we describe the proposed improvements. Finally, in section 5 we experimentally verify the improvements.

2. Stacking Layers are Not Shift-Invariant

In this section, we describe the rationale for having a low-pass filter before the stacking layer. We use a simple illustration to describe the issue in Figure 1. There, we have 6 frames $x[0] \ldots x[5]$ each consisting of a vector of dimension 4. In order to emphasize the effect of aliasing that we will see, the signal

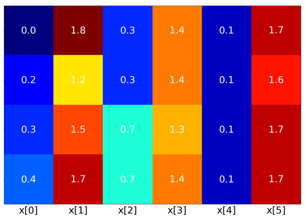

Figure 1: *Example of a signal before a stacking layer.*

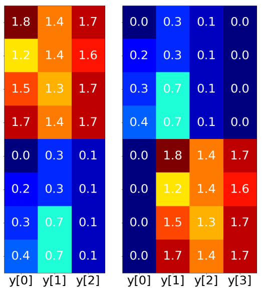

Figure 2: *Example of a signal after a stacking layer without prior shift (left) and with prior shift (right).*

has purposely been made to have a high frequency, alternating between low values for even indices and high values for odd indices.

After the stacking of two consecutive frames (where the frequency reduction parameter is $K = 2$), we have a new signal $y[0] \dots y[2]$ with half as many frames, each having vectors of double the size: 8. This is illustrated on the left side of Figure 2. We see that in this example, the top 4 of the 8 coordinates of each vector have relatively high values, while the bottom 4 have relatively low values. If instead of directly passing the signal x into the stacking layer, we had first shifted it by appending first a vector of zeros, we would get the stacking result of the right side of Figure 2. We can see there that the location of the high and low values in the y vectors are flipped.

Even though the next layer after this stacking step may be permutation-invariant (e.g. by exchanging the weights on the high and low sections), the overall model is still not shift-invariant. Since we want the model to behave equally well with either unshifted or shifted inputs, the shift is not known a-priori and therefore the location of the high values is also not known.

One could be tempted to remediate this effect by averaging the high and low section of the y vectors. However, that would not solve the issue of fractional shifts due to shift in the raw audio signal that are not an integer multiple of the frame sampling rate. The answer should be more refined.

While it thus appears that the stacking layer could theoretically be problematic, in practice, the issue may not be very severe since some works in existing literature ignores this problem [11, 12]. We experimentally show that even though this problem does not occur in matching test conditions, it can potentially degrade performance in mismatched test conditions.

3. Base model

3.1. Architecture

We used the exact model architecture derived from [11, 12], which we briefly summarize here. The input of a model are 128-dimensional log-mel filterbank energies sampled every 32ms with a 10ms hop. This frequency was chosen as a tradeoff between computational load and accuracy [13]. Features from 4 consecutive frames were stacked to create a 512-dimensional input. The target of the model are ground-truth label sequences of word-pieces.

The model is an RNN-T [14, 15] that consists of an encoder, a joint network, and a decoder. We did not use the listen-attend and spell [10] rescoring of [11, 12]. The encoder takes as input log-mel features and computes encoder features through 8 unidirectional long short-term memory (LSTM, [16]) layers of 2,048 hidden units followed by a 640-dimensional projection layer. Between the first and second layers, the frame rate is halved using a stacking layer with $K = 2$. This was motivated by [10, 17] and is explained in detail in section 3.1 of [11]. The goal is to reduce the computational load without degrading the quality of the recognition. This approach is natural, as it is expected that the summarization of the audio signal can be done at a lower rate as we progress through the encoder layer, i.e., as we go from log-mel filterbank to a more abstract representation of the speech at higher encoder layers. However, in the present work, we will revisit the assumption that the stacking layer respects the Nyquist sampling theorem.

The joint layer is a feed forward network with 640 hidden units and the prediction network is an LSTM. The final stage of the model is a softmax with 4,096 units, one for each of our word-piece units. Our base model is the exact same model as in [11, 12] and it has about 120M parameters. We use a forward-backward training algorithm [18] for loss computation; the models are trained on 8x8 Tensorflow Processing Units (TPUs, [19]) with a global batch size of 4,096.

We also used a supervised training set that was similar to the one used by [20, 21], consisting of English utterances. The data was augmented by adding reverberation and varying levels of noise from YouTube [22]. We did not alter the training utterance by changing the pitch or the encoding of the signal.

3.2. Frequency analysis

We first present an analysis that shows whether or not the aliasing problem exists in RNN-T models. Aliasing would not occur if the upper half of the frequencies had very low energy compared to the lower half of the frequencies. It is possible that during the training of the neural network, the weights learned for the first three layers of the encoder naturally produce a signal that will not be aliased when down-sampled.

We thus inspect the frequency components on a donated utterance. For each of the 640 dimensions of x, the input to the stacking layer, we compute the Fourier transform of the time signal, as shown on Figure 3. On the left half of the figure we show a heat map of the frequency response. On the right half of the figure, we show the normalized average frequency response. If the model had learned to perfectly low-pass the signal, we would expect to only see power in the lower half of the frequencies. We do indeed see a downward slope, but the trend is not very strong. This suggests that the frequency response could be improved.

4. Proposed model

4.1. Location and type of the filter

Our proposal is to insert a low-pass filter just before the stacking layer. The filter is designed to remove most of the energy in the upper half of the response, thus alleviating most of the effect of aliasing.

We decided to treat each of the 640 components of the $x[t]$

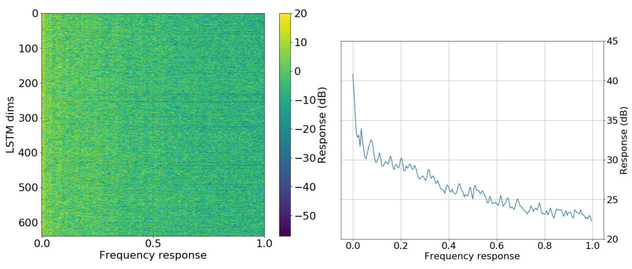
Figure 3: *Frequency analysis of input of the frame stacker in the base model (dB).*

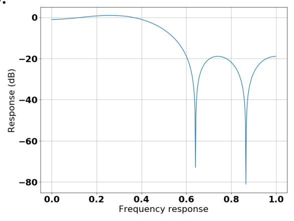
Figure 4: *Low-pass filter frequency response.*

vectors independently. Since there is no temporal dependency between the components, having any kind of mixing between them didn't seem natural. Another way to look at this, we could choose an arbitrary permutation of the 640 components and permute the weights of the model accordingly, and obtain a completely equivalent model.

There is, however, a clear temporal dependency between $x[t]$ and $x[t+1]$, which is the direction in which we want to operate the low-pass filtering. We thus set out to design a 1D filter that would operate on each of the 640 dimensions independently and identically.

4.2. Half-band filter

Contrary to [8], we did not use hand-crafted filters and instead we decided to use standard methods to design the low pass filter [23]. Since we didn't want to introduce a phase distortion between the various 640 components of the signal, we decided to have a symmetric filter [24, 25]. In particular, we settled on a finite impulse response filter with 7 taps designed using the Remez algorithm [26]. We used the open-source Scipy Signal library [27] for computing filter coefficients. Since the stacking parameter was $K = 2$, we wanted to cut off the top half and thus chose two bands. The first band was $[0.0, 0.20]$ where the desired response was 1.0 (keep) and the second band was $[0.30, 0.5]$ the desired response was 0.0 (reject). The Python code was

```
from scipy import signal
h = signal.remez(
    numtaps=7,
    bands=[0.0, 0.20, 0.30, 0.5],
    desired=[1.0, 0.0])
```

As a confirmation that the filter design was appropriate, we plotted its frequency response on Figure 4. We see that, as designed, it removes most upper half of the frequencies.

Throughout our work, we did not change the values of these taps. In particular, during the training of our new model, these weights were fixed from the start to the values above.

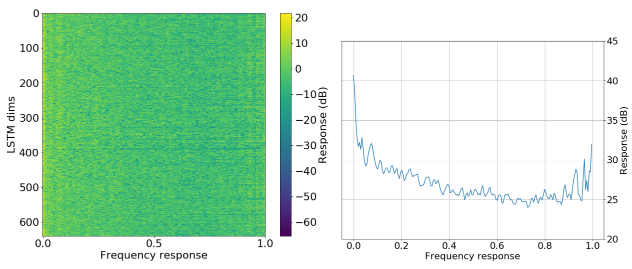
Figure 5: *Frequency analysis of input of the low-pass filter (dB).*

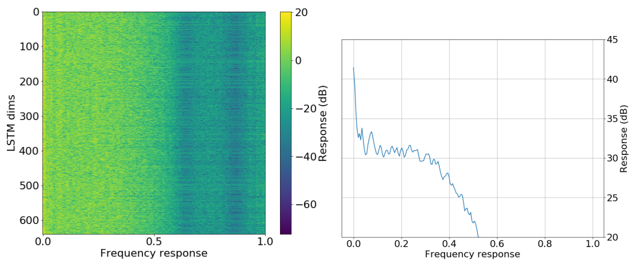
Figure 6: *Frequency analysis of output of the low-pass filter, i.e. the input of the frame stacker (dB).*

4.3. Training of the new model and evaluation

Beside the addition of the low-pass filter before the stacking layer, we did not modify the model in any other way. The weights of the low-pass filter were not tuned and were kept frozen to their initial values. We trained all the other weights from scratch. We trained both the base and proposed models for roughly as many steps as [11, 12].

For evaluation, we set the same beam search parameters for the baseline and proposed models. For simplicity, we did not introduce any contextual biasing, nor did we use an end-pointer model.

4.4. Frequency response before and after the filter

In order to check the effect of our filter on data, we repeated the frequency analysis of section 3.2. Using the same utterance, we looked the frequencies before the low-pass filter and between the low-pass filter and the stacking layer.

We do see in Figure 5 that the frequency distribution before the low-pass filter still contains undesired frequencies. This is not surprising, as the model is free to output any frequency it wants, because they will be removed anyway. After the low-pass filter, in Figure 6, we see a strong attenuation of the undesired frequencies.

5. ASR Results

We measured the ASR quality using word error-rate (WER) on the same voice-search test set as [11, 12]. We compared the base model and the proposed model and the results are shown in Table 1. We see a small improvement in quality of the recognizer when using the proposed model.

	Base WER	Proposed WER	Change
VS	5.8	5.7	-1.7%

Table 1: *Performance of the base and proposed models on a voice-search set.*

We then added white noise to our test sets (Table 2). Even though we had augmented our training data with various types of noise, we see an increased robustness to white noise at various levels. In particular, in the mid-range where the effect on the WER is not extreme, our new model has a somewhat greater relative improvement.

Noise (dBFS)	Base WER	Proposed WER	Change
-72.2	5.8	5.7	-1.7%
-66.2	5.8	5.7	-1.7%
-60.2	5.9	5.7	-3.4%
-54.2	6.1	5.9	-3.3%
-48.2	6.5	6.3	-3.1%
-42.1	7.9	7.8	-1.3%
-38.6	9.7	9.5	-2.1%
-36.1	12.1	11.7	-3.3%
-30.1	21.7	21.3	-1.8%

Table 2: *Performance of the base and proposed models on a voice-search set under various white noise levels in decibels relative to full scale (dBFS).*

Next, we wanted to know whether the new proposed model could perform better under unseen conditions. In particular, some previous work had observed a lack of robustness of ASR models to different codecs [28, 29] when the model is not trained with codec distortion.

Contrary to the previous approaches, we had not augmented the data using different codecs at training time. Instead, we only applied the various codecs at evaluation time. We tested codecs that are frequently applied to lower data usage when contacting a remote server to perform ASR from a cell phone.

To simulate various codecs, we used the open-source software FFMPEG [30]. We report results in Table 3 where the first row (no codec) is a repetition of Table 1. We observe that the performance of the model varies as the code and bitrate used during evaluation differs from those used during training, mirroring the results of [28, 29]. It also appears that as expected the lower the bit-rate the lower the quality of the recognition. The models trained with anti-aliasing filters consistently performs better under these conditions with larger relative improvements when the impact of the codec is more severe.

	Base WER	Proposed WER	Change
No codec	5.8	5.7	-1.7%
OPUS 128K	5.8	5.7	-1.7%
OPUS 24K	5.9	5.8	-1.7%
OPUS 12K	6.3	6.0	-4.8%
MP3 128K	5.9	5.6	-5.1%
MP3 64K	5.9	5.6	-5.1%
MP3 32K	6.1	5.8	-4.9%
MP3 23K	6.5	6.2	-4.6%
MULAW STEREO	5.9	5.6	-5.1%

Table 3: *Performance of the base and proposed models on a voice-search set under various codecs.*

6. Discussion

In essence, our approach is a form of regularization. However, instead of randomly masking some parts of the signal, as done, for example with Dropout [31] or SpecAugment [32], we have a deterministic, first-principle approach. We looked at the architecture of our model and had a deterministic masking of some part of the signal. Because some frequencies do not carry information and are known to cause artifacts in our model architectures, we were able to suppress them without reducing the quality on the original data and improve the power of extrapolation of our model.

The problem of aliasing had already been studied in CNNs and pooling layers, but the problem appears to be present also in stacking layers. Removing the stacking layer, in our case, would have been impractical. It would have negated the speed and size and computation gains of [11, 12]. The solution we proposed is a simple filter that removes the unwanted frequencies. This filter was not trained, because we already knew the desired behavior.

Our solution retains the desirable features of the original model [11, 12]. Because the filter is causal, we can still recognize speech in an online fashion and do not require to see the entire audio sample before processing it. The optimizations that come from dividing the sampling rate by 2 inside the encoder are still fully realizable. The encoder architecture before and after the encoder is left unchanged. Given the tiny size of the filter and that its taps are the same for all 640 dimensions, the computation of the convolution can easily be batched on GPU and TPU.

While the improvements are modest, the impact on the size of the model is negligible. By adding 7 parameters to a model that had 120 million parameters, we leave the model size practically unchanged. Yet, we were able to improve the performance of the model by a modest but noticeable 1.7%.

In addition, we were able to improve the performance on unseen conditions. Without performing data augmentation by using the coders during training, we saw relative improvements of up to 5.1% under these conditions.

7. Conclusions

Frequency aliasing is a very well-studied problem of signal processing. When designing machine learning algorithms, we must consider its effects not only when sampling the input signal but also inside the model itself. CNNs had previously been shown to suffer from aliasing issues; here we focused on stacking layers.

For an experimental validation, we studied the effect of aliasing in an end-to-end speech recognition model. In particular, we focused our attention on the stacking layer that reduces a sampling frequency by 2 inside the encoder of an RNN-T. We first showed that the model had not learned to low-pass the signal, as the sampling frequency was further reduced inside the model.

The analysis of the base model revealed that it had not robustly learned to ignore these high frequencies and was thus susceptible to aliasing. Instead of using a stochastic approach to reducing overfitting, we rely on well known, first-principle theorems from signal processing [6]. By introducing a simple filter, we were able to increase the robustness of our model under conditions unseen at training time.

Given the tiny size of our filter, the size of the model is essentially unchanged. In addition, the performance advantages of reducing the sampling frequency within the model are still conserved. In particular, we are able to compute the output of the different encoder layers in an identical fashion.

8. References

[1] A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*. New York: Prentice-Hall Signal Processing Series, 2009.

[2] K. Turkowski and S. Gabriel, “Filters for common resampling tasks,” *Graphics Gems I. Academic Press*, 1990.

[3] W. Burger and M. Burge, *Principles of Digital Image Processing: Core Algorithms*. New York: Springer, 2009.

[4] V. Saveljev, S.-K. Kim, and J. Kim, “Moiré effect in displays: a tutorial,” *SPIE Optical Engineering*, 2018.

[5] Y. Nishiyama, “Mathematics of fans,” *International Journal of Pure and Applied Mathematics*, vol. 78, 2012.

[6] H. Nyquist, “Certain topics in telegraph transmission theory,” *Transactions of the American Institute of Electrical Engineers*, no. 4, 1928.

[7] Y. Gong and C. Poellabauer, “Impact of aliasing on deep cnn-based end-to-end acoustic models,” *Interspeech 2018*, 2018.

[8] R. Zhang, “Making convolutional networks shift-invariant again,” *ICML*, 2019.

[9] D. Scherer, A. Müller, and S. Behnke, “Evaluation of pooling operations in convolutional architectures for object recognition,” *ICANN*, 2010.

[10] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, “Listen, attend and spell,” *CoRR*, 2015.

[11] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, Q. Liang, D. Bhatia, Y. Shangguan, B. Li, G. Pundak, K. C. Sim, T. Bagby, S. yiin Chang, K. Rao, and A. Gruenstein, “Streaming end-to-end speech recognition for mobile devices,” *ICASSP*, 2019.

[12] T. Sainath, Y. He, B. Li, A. Narayanan, R. Pang, A. Bruguier, S. yiin Chang, W. Li, R. Alvarez, Z. Chen, C. cheng Chiu, D. Garcia, A. Gruenstein, K. Hu, M. Jin, A. Kannan, Q. Liang, I. McGraw, C. Peyser, R. Prabhavalkar, G. Pundak, D. Rybach, Y. Shangguan, Y. Sheth, T. Strohman, M. Visontai, Y. Wu, Y. Zhang, and D. Zhao, “A streaming on-device end-to-end model surpassing server-side conventional model quality and latency,” *ICASSP*, 2020.

[13] G. Pundak and T. Sainath, “Lower frame rate neural network acoustic models,” *Interspeech*, 2016.

[14] A. Graves, “Sequence transduction with recurrent neural networks,” *ArXiv*, vol. abs/1211.3711, 2012.

[15] A. Graves, A. Mohamed, and G. Hinton, “Speech recognition with deep neural networks,” *ICASSP*, 2012.

[16] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Computation*, vol. 9, no. 8, Nov 1997.

[17] R. Alvarez and al., “Introducing the Model Optimization Toolkit for TensorFlow.” [Online]. Available: https://medium.com/tensorflow/introducing-the-model-optimization-toolkit-for-tensorflow-254aca1ba0a3

[18] K. Sim, A. Narayanan, T. Bagby, T. Sainath, and M. Bacchiani, “Improving the Efficiency of Forward-Backward Algorithm using Batched Computation in TensorFlow,” *ASRU*, 2017.

[19] N. P. Jouppi and al., “In-datacenter performance analysis of a tensor processing unit,” *ISCA*, 2017.

[20] A. Narayanan, R. Prabhavalkar, C.-C. Chiu, D. Rybach, T. N. Sainath, and T. Strohman, “Recognizing long-form speech using streaming end-to-end models,” *arXiv preprint arXiv:1910.11455*, 2019.

[21] T. N. Sainath, Y. He, B. Li, A. Narayanan, R. Pang, A. Bruguier, S.-y. Chang, W. Li, R. Alvarez, Z. Chen *et al.*, “A streaming on-device end-to-end model surpassing server-side conventional model quality and latency,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6059–6063.

[22] C. Kim, A. Misra, K. Chin, T. Hughes, A. Narayanan, T. Sainath, and M. Bacchiani, “Generation of large-scale simulated utterances in virtual rooms to train deep-neural networks for far-field speech recognition in google home,” *Interspeech*, 2017.

[23] R. Allred, *Digital Filters for Everyone*. Springer, 2010.

[24] G. Shi, M. Shanechi, and P. Aarabi, “On the importance of phase in human speech recognition,” *IEEE Transactions on Audio, Speech, and Language Processing*, 2006.

[25] J. Fahringer, T. Schrank, J. Stahl, P. Mowlaee, and F. Pernkopf, “Phase-aware signal processing for automatic speech recognition,” *Interspeech*, 2016.

[26] S. B. Davis and P. Mermelstein, “Sur la détermination des polynômes d’approximation de degré donnée,” *Kharkov Mathematical Society*, 1934.

[27] https://docs.scipy.org/doc/scipy/reference/signal.html.

[28] T.-L. Vu, Z. Zeng, H. Xu, and E.-S. Chng, “Audio codec simulation based data augmentation for telephony speech recognition,” *APSIPA ASC*, 2019.

[29] A. Narayanan, A. Misra, K. C. Sim, G. Pundak, A. Tripathi, M. Elfeky, P. Haghani, T. Strohman, and M. Bacchiani, “Toward domain-invariant speech recognition via large scale training,” *Spoken Language Technology Workshop (SLT)*, no. 4, 2018.

[30] https://www.ffmpeg.org.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, 2014.

[32] 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.