On the Use of Pretrained Deep Audio Encoders for Automated Audio Captioning Tasks

Wu, Shih-Lun; Chang, Xuankai; Wichern, Gordon; Jung, Jee-weon; Germain, François G; Le Roux, Jonathan; Watanabe, Shinji

TR2023-141 December 07, 2023

Abstract

Automated audio captioning (AAC) is the task of describing an audio clip that may contain sounds from natural and/or human activities. In the context of autonomous driving, AAC models are a beneficial addition as they can enhance self-driving cars' awareness of the surrounding acoustic environment. Despite being a rather new task, recurring DCASE challenges have sparked continuous research interests in AAC. Following the tremendous successes of pretrained deep models in various fields like computer vision and natural language processing, the majority of DCASE AAC challenge submissions have also used a pretrained deep audio encoder by default. However, not enough exploration and analyses have been done on this vital model component. In this work, we categorize and explain relevant pretraining tasks for deep audio encoders, and compare the downstream AAC performance when using four publicly-accessible pretrained audio encoders in our experiments. We find that, just by altering the pretrained audio encoder in an AAC model, the caption quality metric, SPIDEr-FL, can vary by as much as 15%, and inference speed by more than 100%. Finally, considering the tradeoff between speed and quality, we recommend favoring architecturally simpler audio encoders with more pretraining for time-sensitive applications like self-driving cars.

International Symposium on Future Active Safety Technology toward zero traffic accidents (FAST-zero) 2023

On the Use of Pretrained Deep Audio Encoders for Automated Audio Captioning Tasks

Shih-Lun Wu [1], Xuankai Chang [1], Gordon Wichern [2], Jee-weon Jung [1],
François Germain [2], Jonathan Le Roux [2], Shinji Watanabe [1]

[1] Language Technologies Institute, Carnegie Mellon University, Pittsburgh, PA, USA
{shihlunw, xuankaic, jeeweonj, swatanab}@andrew.cmu.edu
[2] Speech & Audio Team, Mitsubishi Electric Research Labs, Cambridge, MA, USA
{wichern, germain, leroux}@merl.com

Abstract—Automated audio captioning (AAC) is the task of describing an audio clip that may contain sounds from natural and/or human activities. In the context of autonomous driving, AAC models are a beneficial addition as they can enhance self-driving cars' awareness of the surrounding acoustic environment. Despite being a rather new task, recurring DCASE challenges have sparked continuous research interests in AAC. Following the tremendous successes of pretrained deep models in various fields like computer vision and natural language processing, the majority of DCASE AAC challenge submissions have also used a pretrained deep audio encoder by default. However, not enough exploration and analyses have been done on this vital model component. In this work, we categorize and explain relevant pretraining tasks for deep audio encoders, and compare the downstream AAC performance when using four publicly-accessible pretrained audio encoders in our experiments. We find that, just by altering the pretrained audio encoder in an AAC model, the caption quality metric, SPIDEr-FL, can vary by as much as 15%, and inference speed by more than 100%. Finally, considering the tradeoff between speed and quality, we recommend favoring architecturally simpler audio encoders with more pretraining for time-sensitive applications like self-driving cars.

Index Terms—audio captioning, pretraining, PANN, BEATs, seq2seq.

1 Introduction

Automated audio captioning (AAC) [1] is a multimodal task that aims to generate a text description for an input audio clip. The input audios can contain a wide variety of sounds, ranging from natural ones like rain, thunder, or various animal sounds, to those of human activities like car driving, talking, or household chores. Audio captioning systems are highly valuable to the automotive engineering community, as they can help self-driving cars detect, for example, the sirens of fire engines or ambulances early on to allow enough time to make way for them, or the footsteps and chatter of children that may easily fall into the blind spots of a car's cameras to avoid accidents.

Thanks to the rise of deep learning [2], compilation of paired audio-caption datasets [3], [4], and the recurring DCASE challenges on audio captioning [5], [6], [7], [8], AAC has become a fast-developing area of research. Pretrained audio encoders [9], [10], Transformer text decoders [11], [12], data augmentation techniques [13], [14], auxiliary training objectives [15], and reinforcement learning-based optimization [16] have all been experimented with to boost AAC models' performance. However, in our opinion, the choice of pretrained audio encoders is an understudied topic, with most of the DCASE AAC challenge submissions opting for the PANN [9] encoder by default, and only one pivoting to BEATs [10]. On the one hand, the audio encoder's network architecture and pretraining tasks (i.e., tasks that are related to AAC, often with more training data, learned by the audio encoder before AAC) could significantly impact the downstream AAC performance. On the other hand, choosing the right audio encoder for an AAC model is especially crucial to practitioners outside of the deep learning (or deep learning-based audio processing) expertise—as opposed to using various data augmentation techniques and auxiliary training objectives, which requires tedious implementation and tweaks, swapping in and out pretrained audio encoder modules is straightforward to anyone who has learned object-oriented programming, thanks to the easy-to-use text generation pipeline by the open-source HuggingFace library.[1] Therefore, the primary goal of our study is to find out how much impact on AAC performance different audio encoders can cause, using the simplest negative log-likelihood (NLL, read Section 2 for details) training objective for AAC tasks, and only pretrained audio encoders accessible to the public for the best convenience and the least additional carbon footprint incurred.

In this paper, we start by formulating the AAC task as a sequence-to-sequence problem (Section 2), and then describe and compare common pretraining tasks for the audio encoder in AAC models (Section 3). In Section 4, we detail our experimental setup with 4 different versions of the PANN and BEATs audio encoders, and the metrics used to examine caption quality and generation speed. The results are presented and discussed in Section 5, where we also offer our suggestions for the safety-critical use case of self-driving cars.

2 Problem Formulation

The task of automated audio captioning (AAC) can naturally be formulated as a sequence-to-sequence (i.e., seq2seq) [17] problem. The input $\boldsymbol{x}$ is a sequence of real numbers representing an audio waveform typically sampled at 16 to 48 kHz:

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_T);\ x_t \in \mathbb{R}\,, \tag{1}$$

1. github.com/huggingface/notebooks/blob/main/examples/language_modeling_from_scratch.ipynb

where T is the number of waveform samples. The output $\boldsymbol{y}$, i.e., the desired caption for the input $\boldsymbol{x}$, is a sequence of text tokens:

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_N);\ y_n \in \mathbb{N} \cap [1, |\mathcal{V}|] \,, \tag{2}$$

where N is the length of the caption (in number of tokens) and $\mathcal{V}$ is a predefined vocabulary of text tokens.[2]

It is assumed that audio captions in the real world are generated from a conditional probability distribution $p(\boldsymbol{y} \,|\, \boldsymbol{x})$, which, by the chain rule of probability, can be decomposed as:

$$\begin{aligned} p(\boldsymbol{y} \,|\, \boldsymbol{x}) &= p(y_1, y_2, \ldots, y_N \,|\, \boldsymbol{x}) && (3)\\ &= p(y_1|\boldsymbol{x}) p(y_2|y_1; \boldsymbol{x}) \cdots p(y_N|\boldsymbol{y}_{1:N-1}; \boldsymbol{x}) && (4)\\ &= \prod_{n=1}^{N} p(y_n|\boldsymbol{y}_{1:n-1}; \boldsymbol{x}) \,, && (5) \end{aligned}$$

where $\boldsymbol{y}_{1:n-1}$ is a slice of $\boldsymbol{y}$ containing the first $n-1$ tokens, and $\boldsymbol{y}_{1:0} = \varnothing$, which indicates an empty string.

Under the seq2seq framework, an AAC model typically consists of an audio encoder and a text decoder. The audio encoder $f_\phi(\boldsymbol{x})$, implemented by a neural network with parameters ϕ, transforms the waveform samples $\boldsymbol{x}$ into a sequence of (learned) audio features $\boldsymbol{H}^{(\boldsymbol{x})}$:[3]

$$\boldsymbol{H}^{(\boldsymbol{x})} = f_\phi(\boldsymbol{x}) = (\boldsymbol{h}_1^{(\boldsymbol{x})}, \boldsymbol{h}_2^{(\boldsymbol{x})}, \ldots, \boldsymbol{h}_{T'}^{(\boldsymbol{x})});\ \boldsymbol{h}_{t'}^{(\boldsymbol{x})} \in \mathbb{R}^d \,, \tag{6}$$

where d is the preset output dimension of the audio encoder, and T' is the number of encoded audio features (usually $T' \ll T$). The text decoder, implemented by another neural network with parameters θ, then takes the encoded audio features $f_\phi(\boldsymbol{x})$ and a partially complete caption $\boldsymbol{y}_{1:n-1}$ to estimate the probability distribution over $|\mathcal{V}|$:

$$p_\theta(y_n = y \,|\, \boldsymbol{y}_{1:n-1}; f_\phi(\boldsymbol{x})),\ \forall y \in \mathcal{V} \,, \tag{7}$$

which in turn, by the decomposition in Eqs. (3)-(5), can estimate the true distribution of audio captions $p(\boldsymbol{y} \,|\, \boldsymbol{x})$.

The AAC models are most commonly trained via maximum likelihood estimation (MLE) [18], i.e., searching for the best parameters $\{\phi, \theta\}$ through the following optimization:

$$\underset{\phi,\theta}{\arg\max}\, \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} \left[p_\theta(\boldsymbol{y} \,|\, f_\phi(\boldsymbol{x})) \right] , \tag{8}$$

where $\mathcal{D}$ is the training data distribution. In practice, to obtain better numerical stability and to follow the convention of framing an optimization problem as a minimization, the negative log-likelihood is more often used as the loss function, leading to the following optimization equivalent to Eq. (8):

$$\underset{\phi,\theta}{\arg\min}\, \mathbb{E}_{(\boldsymbol{x},\boldsymbol{y})\sim\mathcal{D}} \left[-\log p_\theta(\boldsymbol{y} \,|\, f_\phi(\boldsymbol{x})) \right] . \tag{9}$$

The optimization can be done using stochastic gradient descent (SGD) [19] algorithms, where the parameters $\{\phi, \theta\}$ are iteratively updated using the gradient computed on sampled data instances, i.e., $\nabla_{\phi,\theta} - \log p_\theta(\boldsymbol{y} \,|\, f_\phi(\boldsymbol{x}))$.

2. Every integer in $\mathbb{N} \cap [1, |\mathcal{V}|]$ gets mapped either to a word (e.g., *bird*) or to a part of a word (e.g., the suffix *-ing*) by the vocabulary $\mathcal{V}$.
3. The superscript $^{(\boldsymbol{x})}$ is used to clarify that $\boldsymbol{H}^{(\boldsymbol{x})}$ originates from $\boldsymbol{x}$.

3 Pretraining the Audio Encoder

The goal of *pretraining* the audio encoder $f_\phi(\boldsymbol{x})$ is to equip it with relevant background knowledge before it is *finetuned* on the AAC task as detailed in the previous section, so that it can perform well with less amount of AAC training data. Relevant background knowledge includes, for example, the correlation between samples in an audio clip and the interplay between audio and language, which reside in a wider range of data beyond carefully curated audio-caption pairs. In the following subsections, we will introduce three pretraining tasks (and the required data formats) in the order of increasing supervision from language, and provide a comparison between them in the end.

3.1 Masked Audio Language Modeling (MALM)

The idea of MALM originates from masked language modeling (MLM) in text domain [20]. It is a self-supervised[4] learning method which randomly masks out some parts of a sequence and trains the model to infer the masked parts from the unmasked ones. This induces the model to learn co-occurrence patterns of signals.

The MALM approach in audio domain was proposed by Chen et al. [10], and it requires an additional audio tokenizer $e_\psi(\boldsymbol{x})$, a neural network with parameters ψ, to discretize/compress audio waveform $\boldsymbol{x} \in \mathbb{R}^T$ (cf. Eq. (1)) into a token sequence $\boldsymbol{c} \in \mathbb{N}^{T'}$:

$$e_\psi(\boldsymbol{x}) = \boldsymbol{c}^{(\boldsymbol{x})} = (c_1^{(\boldsymbol{x})}, c_2^{(\boldsymbol{x})}, \ldots, c_{T'}^{(\boldsymbol{x})});\ c_{t'}^{(\boldsymbol{x})} \in \mathbb{N} \cap [1, |\mathcal{C}|] \,, \tag{10}$$

where $\mathcal{C}$ is the vocabulary of discretized audio tokens (commonly called the *codebook*). Note that the length of $\boldsymbol{c}$ is the same as that of $\boldsymbol{H}^{(\boldsymbol{x})}$, the output features of the audio encoder (cf. Eq. (6)).[5]

The MALM pretraining task is constructed as follows. First, the waveform $\boldsymbol{x}$ is internally converted into a mel spectrogram[6] and split into T' patches before being processed by learned parameters in f_ϕ and e_ψ. Let the mel spectrogram patches be $\boldsymbol{S}^{(\boldsymbol{x})} \in \mathbb{R}^{T' \times k}$, where k is the number of elements in a patch, a random set of indices $\mathcal{M} \subset \{1, 2, \ldots, T'\}$ is chosen to mask out some patches in $\boldsymbol{S}^{(\boldsymbol{x})}$. The encoder f_ϕ is asked to predict the tokens corresponding to the masked patches based on the unmasked ones, with the help of a linear classifier g_λ (i.e., a linear transformation + a softmax operation) attached to its output features $\boldsymbol{H}^{(\boldsymbol{x})}$. Mathematically, f_ϕ and g_λ produce a probability distribution over $\mathcal{C}$ at all masked indices $t' \in \mathcal{M}$:

$$p_{\phi,\lambda}(c_{t'}^{(\boldsymbol{x})} = c \,|\, \boldsymbol{S}_{\mathcal{M}^c}^{(\boldsymbol{x})}),\ \forall c \in \mathcal{C} \,, \tag{11}$$

where $\boldsymbol{S}_{\mathcal{M}^c}^{(\boldsymbol{x})}$ denotes the unmasked patches of $\boldsymbol{S}^{(\boldsymbol{x})}$. Similar to Eq. (9), the optimization problem can hence be written as:

$$\underset{\phi,\lambda}{\arg\min}\, \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}_{\text{pt}}} \Big[\sum_{t' \in \mathcal{M}} -\log p_{\phi,\lambda}(c_{t'}^{(\boldsymbol{x})} \,|\, \boldsymbol{S}_{\mathcal{M}^c}^{(\boldsymbol{x})}) \Big] , \tag{12}$$

where $\mathcal{D}_{\text{pt}}$ denotes the pretraining data distribution.

3.2 Multi-Label Audio Classification (MLAC)

The MLAC method trains the model to recognize the collection of events occuring in an audio clip. This was made possible largely by the AudioSet [21] dataset and its tree-structured class

4. I.e., a learning paradigm which does not require human-labeled data.
5. In [10], the audio tokenizer e_ψ is learned alternately with the encoder f_ϕ; to keep only the key idea here, we refer readers to their paper for more details.
6. A representation showing energy levels on the time-frequency plane.

ontology,[7] which led to the standardized task of *audio tagging* addressed by numerous subsequent research works [9], [22], [23].

Since the audio event classes are not mutually exclusive (e.g., an audio can contain the sound of a flying *aircraft* and falling *rain*), the task to be learned is multi-label classification, i.e., doing binary classification on every class. Let the set of classes be $\mathcal{C}$. To perform the task, typically, the output features $\boldsymbol{H}^{(\boldsymbol{x})}$ are first mean-pooled across the time dimension before being passed to g_λ, a family of $|\mathcal{C}|$ binary classifiers, to output the probabilities of each class $c \in \mathcal{C}$ being positive in the audio, i.e.,

$$\bar{\boldsymbol{h}}^{(\boldsymbol{x})} = \text{MeanPool}_{t'}\{\boldsymbol{h}_1^{(\boldsymbol{x})}, \dots, \boldsymbol{h}_{T'}^{(\boldsymbol{x})}\};\ \bar{\boldsymbol{h}}^{(\boldsymbol{x})} \in \mathbb{R}^d\,, \tag{13}$$

$$\boldsymbol{o}^{(\boldsymbol{x})} = (p_{\phi,\lambda}(c \,|\, \boldsymbol{x}))_{c\in\mathcal{C}} = g_\lambda(\bar{\boldsymbol{h}}^{(\boldsymbol{x})});\ \boldsymbol{o}^{(\boldsymbol{x})} \in [0,1]^{|\mathcal{C}|}\,, \tag{14}$$

where g_λ is composed of a linear transformation followed by an element-wise sigmoid function to squash outputs into the interval $[0, 1]$. The optimization objective here is therefore:

$$\arg\min_{\phi,\lambda} \mathbb{E}_{(\boldsymbol{x},\boldsymbol{v})\sim\mathcal{D}_{\text{pt}}}[-\sum_{c\in\mathcal{C}}(v_c \log o_c^{(\boldsymbol{x})} + (1-v_c)(1-\log o_c^{(\boldsymbol{x})}))]\,, \tag{15}$$

where $\boldsymbol{v} \in \{0,1\}^{|\mathcal{C}|}$ is the ground truth associated with $\boldsymbol{x}$, i.e., v_c is 1 when class c is positive, and 0 when negative. In the literature, this objective function is often called *binary cross-entropy*.

3.3 Audio-Text Contrastive Learning (ATCL)

ATCL aims to learn cross-modal representations using separate unimodal (i.e., audio and text) encoders. By aligning the unimodal encoders' representations, concepts in the language can be infused into the deep audio features $\boldsymbol{H}^{(\boldsymbol{x})}$, which is highly beneficial to language-related downstream tasks like audio captioning. This technique was first pioneered on image-text paired data [24] and later brought to the audio-text domain [25], [26].

To implement ATCL, we need paired audio-text data $(\boldsymbol{x}, \boldsymbol{y})$, and an (often already pretrained) text encoder $e_\psi(\boldsymbol{y})$ to convert the text $\boldsymbol{y}$ (as a token sequence, cf. Eq. (7)) into a high-dimensional feature $\boldsymbol{h}^{(\boldsymbol{y})} \in \mathbb{R}^d$.[8] On the audio encoder side, the pooled feature $\bar{\boldsymbol{h}}^{(\boldsymbol{x})}$ (see Eq. (13)) is used as a summarized audio representation. Then, learning is driven by the InfoNCE [15] loss function, which is computed across data instances in a sampled batch $\mathcal{B} \subset \mathcal{D}_{\text{pt}}$:

$$\text{sim}(\bar{\boldsymbol{h}}^{(\boldsymbol{x})}, \boldsymbol{h}^{(\boldsymbol{y})}) = \exp\big(\frac{\bar{\boldsymbol{h}}^{(\boldsymbol{x})\top}\boldsymbol{h}^{(\boldsymbol{y})}}{||\bar{\boldsymbol{h}}^{(\boldsymbol{x})}||\,||\boldsymbol{h}^{(\boldsymbol{y})}||} \cdot \frac{1}{\tau}\big), \tag{16}$$

$$\mathcal{L}_{\text{NCE\_x}} = \mathbb{E}_{\mathcal{B}\subset\mathcal{D}_{\text{pt}}}\Big[\sum_{i=1}^{|\mathcal{B}|} -\log \frac{\text{sim}(\bar{\boldsymbol{h}}_i^{(\boldsymbol{x})}, \boldsymbol{h}_i^{(\boldsymbol{y})})}{\sum_j^{|\mathcal{B}|}\text{sim}(\bar{\boldsymbol{h}}_j^{(\boldsymbol{x})}, \boldsymbol{h}_i^{(\boldsymbol{y})})}\Big], \tag{17}$$

$$\mathcal{L}_{\text{NCE\_y}} = \mathbb{E}_{\mathcal{B}\subset\mathcal{D}_{\text{pt}}}\Big[\sum_{i=1}^{|\mathcal{B}|} -\log \frac{\text{sim}(\bar{\boldsymbol{h}}_i^{(\boldsymbol{x})}, \boldsymbol{h}_i^{(\boldsymbol{y})})}{\sum_j^{|\mathcal{B}|}\text{sim}(\bar{\boldsymbol{h}}_i^{(\boldsymbol{x})}, \boldsymbol{h}_j^{(\boldsymbol{y})})}\Big], \tag{18}$$

$$\mathcal{L}_{\text{NCE}} = \frac{1}{2}(\mathcal{L}_{\text{NCE\_x}} + \mathcal{L}_{\text{NCE\_y}})\,, \tag{19}$$

where i and j index instances in a same batch, and $\text{sim}(\cdot,\cdot)$ measures the similarity between two representations, which can be sharpend (or flattened) by setting a lower (or higher) temperature τ. Following the above, the optimization objective is simply:

$$\arg\min_{\phi} \mathcal{L}_{\text{NCE}}\,. \tag{20}$$

Intuitively, when only the audio encoder f_ϕ is being trained,[9] the InfoNCE loss components (i.e., Eqs. (17) and (18)) are pulling an audio feature $\bar{\boldsymbol{h}}_i^{(\boldsymbol{x})}$ closer to its paired text feature $\boldsymbol{h}_i^{(\boldsymbol{y})}$ while pushing other audio features $\bar{\boldsymbol{h}}_{j,(j\neq i)}^{(\boldsymbol{x})}$ away from $\boldsymbol{h}_i^{(\boldsymbol{y})}$. Through such optimization over the entire dataset $\mathcal{D}_{\text{pt}}$, under the limited capacity of the $\mathbb{R}^d$ space, the audio encoder is encouraged to put audios with similar language concepts in the same neighborhood in $\mathbb{R}^d$, and place the most dissimilar ones the furthest apart.

3.4 Comparison

In terms of the *knowledge* obtained, only MALM forces the audio encoder to learn fine-grained, time-varying features, which are especially helpful in cases where multiple events occur successively in the audio, e.g., "*a car drives by followed by people chattering on the street*," or where the sound is gradually evolving, e.g., "*the ambulance siren gets louder and louder as it approaches*." On the other hand, although MLAC and ATCL apply their loss function only on a single feature pooled through the temporal dimension, they are more closely related to audio captioning as they teach the audio encoder to link up its representations with concepts in the texts paired with audios, with ATCL allowing richer and more flexible text inputs, i.e., sentences with nouns, adjectives, and temporal/positional relations, as opposed to combinations of predefined sound classes in MLAC.

Regarding the *data* available for pretraining, MALM is the one with the least restrictions since it does not need any annotation. MLAC requires the most careful data curation to ensure annotated labels are consistent with its class ontology. ATCL has a flexibility in between MALM and MLAC. Although text inputs are required, less accurate descriptions of the audio (compared to AAC training data) are acceptable, so we can leverage the vast amount of paired audio and text uploaded to the web, where an audio may be simply described as, for example, "*upbeat jazz music*." Even MLAC training data can be used for ATCL by just writing out the labels, e.g., "*the sound of [class 1], [class 2], . . . , and [class C]*." Both of the aforementioned data collection measures were used in [26].

Training-wise, MLAC is arguably the most straightforward as it only involves the audio encoder and there are no hyperparameters to tune in the objective function. The only detail that requires attention is balanced sampling [22], which prevents the model from collapsing to only predicting the most common classes. ATCL is fairly easy as well if we keep the text encoder frozen. It will be helpful to pick a strong text encoder by referring to, for example, the Massive Text Embedding Benchmark [27], and to experiment with several temperature values τ in the InfoNCE loss function. MALM pretraining is the trickiest since there are two models to be trained alternately. We have to ensure that the audio tokenizer $e_\psi(\boldsymbol{x})$ sufficiently utilizes the codebook $\mathcal{C}$ [28], and also carefully select the fraction of audio signal masked by $\mathcal{M}$, such that the reconstruction task is neither too trivial nor too difficult for the audio encoder $f_\phi(\boldsymbol{x})$ to learn.

To *evaluate* the pretrained audio encoder, for MLAC, we can simply measure its auto tagging (i.e., multi-label classification)

7. E.g., *human voice* and *laughter* are two classes in AudioSet, with *laughter* being in the subtree of *human voice*.

8. For simplicity of formulation, here we assume that the learned audio and text features have the same dimension *d*. The dimensions can easily be adjusted by a linear transformation if they are different.

9. We note that it is better not to update the text encoder e_ψ when it has been pretrained on a much larger volume of web text.

performance on the evaluation set. For ATCL, it is also straightforward to evaluate it on text-audio retrieval, which computes the similarity between the audio representation and those of a pool of text candidates to select the most relevant text description for an audio clip. On the contrary, evaluating a MALM-pretrained audio encoder is not directly possible, as the accuracy on predicting masked audio tokens is barely interpretable. We can attach additional network components on top of the pretrained encoder to test it out on downstream tasks like audio tagging, but doing so entails an extra finetuning process.

Despite the individual advantages and drawbacks discussed, we note that multiple pretraining methods can be applied at the same time to enjoy the best of all methods, as long as all necessary network components are attached to the audio encoder. Such a strategy was used in some of the pretrained audio encoders we experiment with, which will be covered in the next section.

4 Experiments

4.1 Datasets

The main dataset we use for finetuning and evaluation on the automated audio captioning (AAC) task is Clotho [4],[10] which consists of 6K audio clips (about 4K / 1K / 1K in training / validation / evaluation splits respectively), each being 15$\sim$30 seconds long and paired with 5 human-annotated captions. During Clotho's data collection, the authors ensured that no information is leaked to the captions from video clips associated with the audios, restricted caption length to 8$\sim$20 words, and maintained the consistency of vocabulary distribution in the 3 splits. Since the Clotho dataset itself is small in size, following common practice in DCASE AAC challenges [5], [6], [7], we additionally leverage the AudioCaps dataset [3], which contains 45K paired audios and captions (audios are 10-second long and sampled from AudioSet [21]), to finetune our models on it before Clotho.

As for pretraining, all publicly available audio encoder implementations [9], [10] and model weights we experiment with utilized AudioSet [21], which comprises 2M clips and 5K hours of audio (i.e., 40$\times$ and 130$\times$ the size of AudioCaps and Clotho respectively), as their large-scale pretraining dataset.

4.2 Pretrained Audio Encoders

We leverage two publicly released, pretrained audio encoders, PANN [9] and BEATs [7], as the audio encoder $f_\phi(\boldsymbol{x})$ in our audio captioning models. The major difference between the two in network architecture is that PANN is based on convolutional neural network (CNN) layers [29], assuming locality and translation invariance of input data and applying (a large number of) learnable filter windows that 'slide' over the entire input to extract features from neighboring input elements. BEATs, on the other hand, has Transformer [11] layers as its backbone. Instead of being restricted to the neighborhood, every element in the input is allowed to access and gather information from arbitrary parts of the entire input through the Transformer's *self-attention* mechanism, making Transformers a more general and stronger sequence processor.

The PANN and BEATs models (2 versions for each) we use represent 4 combinations of pretraining tasks: PANN is either pretrained on MLAC alone,[11] or on MLAC+ATCL.[12] BEATs is either pretrained on MALM only,[13] or on MALM+MLAC.[14]

10. Version 2.1, as available at `zenodo.org/record/4783391`.

11. The `Cnn14_16k_mAP=0.438.pth` checkpoint on `zenodo.org/record/3987831`.

12. Checkpoint provided in DCASE 2023 AAC challenge official baseline: `zenodo.org/record/7752975`. ATCL is learned on Clotho dataset.

4.3 Text Decoder

To all 4 audio encoders we include in our experiments, we attach the BART [12] Transformer text decoder, which learns $p_\theta(y_n \,|\, \boldsymbol{y}_{1:n-1}; f_\phi(\boldsymbol{x}))$ (cf. Eq. (7)) to generate captions. The BART decoder shares a nearly identical attention mechanism with the BEATs audio encoder, but in addition to *self-attending* to the partially completed caption text tokens, it *cross-attends* to the sequence of audio features produced by the encoder f_ϕ, to access and summarize the information in the audio. We utilize the open-source HuggingFace BART implementation[15] (and hence its text generation pipeline) and train the BART weights from scratch.[16]

During inference, we use the beam search algorithm [30] implemented in HuggingFace `model.generate()` method[17] and set the beam width to 4, which greedily keeps 4 most likely (determined by the log-probabilities $\sum_n \log p_\theta(y_n| \dots)$) partial captions at any given time, and finally returns the most likely one as the estimated caption.

4.4 Training (Finetuning) Details

Across all of our settings (i.e., 4 audio encoders), we finetune our AAC models first for 10 epochs on the AudioCaps [3] dataset, then for 100 epochs on the Clotho [4] dataset.[18] The objective function used to optimize the models was detailed in Eq. (9). We apply the AdamW optimizer [31], which adaptively adjusts the step size of model weight updates based on gradient history, and automatically decays model weights to mitigate overfitting.

Some important hyperparameters used during finetuning are: batch size = 32; base learning rate = 2×10^{-4} for the AudioCaps stage, and 2×10^{-5} for the Clotho stage.

4.5 Evaluation Metrics

We compute various metrics to examine the quality of generated captions, namely, BLEU, ROUGE-L, METEOR, CIDEr, SPICE, SPIDEr, and SPIDEr-FL, which compare model generations to human-written references in different ways.

BLEU [32] is a precision-based metric, which calculates the percentage of n-grams (i.e., substrings of n words) that also appears in real data. We use BLEU-4 here, which computes the score over 4-grams. ROUGE-L [33], on the other hand, aligns more with the concept of recall. It computes the ratio between the length of the *longest common subsequence* between generated and reference texts, and the reference text's length. METEOR [34] not only considers the harmonic mean of n-gram precision and recall, but also penalizes text fragmentation that occurs when aligning generated and reference texts word-to-word. CIDEr [35] and SPICE [36] were particularly developed for image captioning [37] tasks, and were generally found to correlate better with human perception of quality than BERT, ROUGE-L, and METEOR. Similar

13. The `BEATs_iter3+ (AS2M)` checkpoint on `github.com/microsoft/unilm/tree/master/beats`.

14. The `Fine-tuned BEATs_iter3+ (AS2M) (cpt1)` checkpoint on `github.com/microsoft/unilm/tree/master/beats`.

15. `huggingface.co/docs/transformers/model_doc/bart`

16. This is due to the fact that pretrained BART models cross-attend to encoded *text* features, not *audio* features.

17. `huggingface.co/docs/transformers/main/main_classes/text_generation`.

18. An *epoch* is a full pass through all samples in the dataset.

TABLE 1: Characteristics and performance (on Clotho [4] evaluation split) of audio captioning models with different pretrained audio encoders. All metrics are the higher the better. All systems use BART Transformer [12] text decoder. Acronyms for pretraining tasks—MALM: masked audio language modeling; MLAC: multi-label audio classification; ATCL: audio-text contrastive learning.

	Pretraining tasks			**Captioning quality metrics**						
Audio encoder f_ϕ (backbone)	MALM	MLAC	ATCL	BLEU-4	ROUGE-L	METEOR	CIDEr	SPICE	SPIDEr	SPIDEr-FL
PANN [9] (CNN)	✗	✓	✗	.148	.366	.170	.387	.117	.252	.248
	✗	✓	✓	.154	.371	.173	.415	.123	.269	.263
BEATs [10] (Transformer)	✓	✗	✗	.156	.375	.176	.406	.122	.264	.261
	✓	✓	✗	**.170**	**.383**	**.184**	**.448**	**.129**	**.289**	**.287**

TABLE 2: Tradeoff in captioning quality and inference speed, affected by feature frame rate, feature downsampling factor, and audio encoder backbone. The BEATs encoder here uses MALM+MLAC pretraining tasks, while the PANN uses MLAC+ATCL. **RTF** stands for **real time factor**, which means how many seconds of audio a model can process in one second of wall-clock time.

	Feature Characteristics		**Inference speed (RTF)**			**Captioning quality metrics**					
Audio encoder f_ϕ	Downsample×	Frame rate	Mean	Median	10th %tile	BLEU-4	ROUGE-L	METEOR	CIDEr	SPICE	SPIDEr-FL
BEATs [10]	1	50	42.9	45.3	33.5	**.170**	.383	**.184**	**.448**	**.129**	**.287**
	2	25	45.7	47.4	42.7	.165	**.384**	.183	.434	.128	.279
	4	13	46.5	47.8	43.6	.162	.379	.183	.423	.126	.273
	8	6	46.5	47.7	43.2	.164	.378	.180	.420	.122	.269
PANN [9]	1	1	**104.6**	**108.4**	**81.9**	.154	.371	.173	.415	.123	.263

to BERT and METEOR, CIDEr also measures n-gram overlap, but it upweights more informative words and downweights general ones. For example, in "*rain splashes on the ground*," the word *splashes* could be the most informative, and *the* or *is* could be the most general/unimportant ones. Meanwhile, SPICE focuses on the overlap computed on semantic graphs constructed by objects, object attributes, and relations. SPIDEr [38], which is the simple mean of CIDEr and SPICE, had been the official evaluation metric in DCASE AAC challanges until 2022. SPIDEr-FL [39] became the DCASE official metric in 2023. It is largely the same as SPIDEr, but requires training an additional binary classifier [40] to recognize and penalize disfluent generations.

In addition to quality, how quickly the model can generate captions is also a crucial aspect, especially in life-critical use cases like self-driving cars. Therefore, we also compute the real-time factor (RTF) during inference, which is defined as the ratio of the duration of the input audio to the duration taken by the model to generate a caption for it. For instance, an RTF of 40 means the model can generate captions for 40 seconds of audio in 1 second.

5 Results and Discussion

Table 1 displays our models' performance metrics on the Clotho [4] evaluation split (with 1,045 samples in total). Holding the text decoder (i.e., BART [12]) and inference search algorithm (i.e., beam search [30]) constant, using a BEATs Transformer [10] that was pretrained on both masked audio language modeling (MALM) and multi-label audio classification (MLAC) as the audio encoder f_ϕ leads to the overall best performance. On SPIDEr-FL, it outperforms the worst setting (PANN [9] with only MLAC pretraining) by 3.9 percentage points, or 15% relatively. If we pay attention to different pretraining settings for the same encoder architecture (i.e., first 2 rows for PANN, and last 2 rows for BEATs), we can see that additional pretraining tasks help with audio captioning performance in general: adding the audio-text contrastive learning (ATCL) task gives PANN an extra 1.5 percentage points on SPIDEr-FL, while applying MLAC on top of MALM in BEATs leads to an even larger improvement of 2.6 SPIDEr-FL points. We encourage readers to check out some of the captions generated by our models.[19]

However, here, we note that direct comparison across BEATs and PANN encoders involved in our experiments is not fair due to their stark differences in network architecture and time resolution of output audio features (50 frames/sec for BEATs, and only $\sim$1 frames/sec for PANN). The fact that there is not a pair of publicly released PANN and BEATs pretrained on the same task combination makes the comparison even more challenging, as we cannot isolate how much advantage comes from BEATs's Transformer architecture and higher-resolution features. Meanwhile, it is reasonable to limit our experiments to readily available pretrained models, as pretraining on large-scale datasets requires a tremendous amount of compute resources, and also deep expertise in audio machine learning, which practitioners who just need to build an audio captioning application might not have.

Therefore, to understand how much high-resolution features help with BEATs' performance, we attempt to downsample BEATs output audio features (i.e., $\boldsymbol{H}^{(\boldsymbol{x})}$, cf. Eq. (6)) by 2$\sim$8 times, which can be done with just one line of array striding code,[20] before the BART decoder cross-attention, to compare them with the better PANN setup (i.e., with both MLAC and ATCL pretraining). This helps us know how much the performance would drop with lower-resolution audio features from BEATs. Additionally, to get a sense of how much extra time BEATs takes despite its better performance, we also check the real-time factor (RTF) in this set of comparison. The results are shown in Table 2. We can see that BEATs's performance degrades steadily with more aggressive downsampling, with the 8x-downsampled setting being just 0.6 percentage point (on SPIDEr-FL) ahead of PANN, which still outputs encoded audio features at the much lower resolution. Focusing on inference speed, PANN is more than twice as fast as any BEATs version, likely due to its more simple network architecture and lower feature resolution. Downsampling BEATs features does not accelerate the model by much as it does not

19. Generated samples: `https://bit.ly/3DZJBFa`.
20. e.g., in Python, `arr = arr[::2]`, works as 2x downsampling.

reduce the computation within BEATs audio encoder, but only those in BART decoder cross-attention.

Given that models achieving a SPIDEr-FL score over 0.25 are able to generate captions that accurately describe input audios in most cases, we suggest that developers who are building time-sensitive audio captioning systems opt for the faster PANN audio encoder. Moreover, systematic testing measures for deep learning systems [41] should be incorporated to ensure that the model returns correct captions in time in critical scenarios. Taking the self-driving car use case as an example, important test cases may be detecting a fast-approaching ambulance, and children playing or running around the car.

6 Conclusion

In this paper, we first formulated the automated audio captioning problem (AAC) as estimating a conditional probability distribution, which can be learned with sequence-to-sequence (seq2seq) neural network models. Then, we described 3 relevant pretraining tasks for the audio encoder in seq2seq AAC models that may leverage a wider range of data and improve the downstream audio captioning performance. These tasks are: masked audio language modeling (MALM), multi-label audio classification (MLAC), and audio-text contrastive learning (ATCL), in the order of increasing supervision from text. In our experiments, we finetuned 2 publicly available audio encoders, i.e., PANN and BEATs, with a total of 4 pretraining task combinations. Considering both captioning performance and inference speed, we recommended that practitioners choose the faster PANN encoder pretrained on more tasks for time-critical use cases like self-driving cars.

As a concluding note, our work specifically attempted to elucidate the impact on audio captioning performance caused by different pretrained audio encoders, and hence only used the basic negative log-likelihood loss function for AAC finetuning. Readers should be aware that utilizing auxiliary loss functions and various data augmentation techniques [7], [8] during AAC finetuning is an active area of research, and has achieved SPIDEr-FL scores over 0.30 without causing significant inference-time overhead.

Acknowledgments

Shih-Lun Wu, Xuankai Chang, Jee-weon Jung, and Shinji Watanabe used the Bridges2 system at PSC and Delta system at NCSA through allocation CIS210014 from the Advanced Cyberinfrastructure Coordination Ecosystem: Services & Support (ACCESS) program, which is supported by National Science Foundation grants #2138259, #2138286, #2138307, #2137603, and #2138296.

References

[1] X. Mei, X. Liu, M. D. Plumbley, and W. Wang, "Automated audio captioning: an overview of recent progress and new challenges," *EURASIP Journal on Audio, Speech, and Music Processing*, 2022.

[2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, 2015.

[3] C. D. Kim, B. Kim, H. Lee, and G. Kim, "AudioCaps: Generating captions for audios in the wild," in *Proc. NAACL-HLT*, 2019.

[4] K. Drossos, S. Lipping, and T. Virtanen, "Clotho: an audio captioning dataset," in *Proc. ICASSP*, 2020.

[5] W. Yuan, Q. Han, D. Liu, X. Li, and Z. Yang, "The DCASE 2021 challenge task 6 system: Automated audio captioning with weakly supervised pre-training and word selection methods," DCASE2021 Challenge, Tech. Rep., 2021.

[6] X. Xu, Z. Xie, M. Wu, and K. Yu, "The SJTU system for DCASE2022 challenge task 6: Audio captioning with audio-text retrieval pre-training," DCASE2022 Challenge, Tech. Rep., 2022.

[7] S.-L. Wu, X. Chang, G. Wichern, J.-w. Jung, F. Germain, J. Le Roux, and S. Watanabe, "BEATs-based audio captioning model with Instructor embedding supervision and ChatGPT mix-up," DCASE2023 Challenge, Tech. Rep., 2023.

[8] J.-H. Cho, Y.-A. Park, J. Kim, and J.-H. Chang, "HYU submission for the dcase 2023 task 6a: automated audio captioning model using AL-MixGen and synonyms substitution," DCASE2023 Challenge, Tech. Rep., 2023.

[9] 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*, 2020.

[10] S. Chen, Y. Wu, C. Wang, S. Liu, D. Tompkins, Z. Chen, and F. Wei, "BEATs: Audio pre-training with acoustic tokenizers," *arXiv preprint arXiv:2212.09058*, 2022.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. NeurIPS*, 2017.

[12] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *Proc. ACL*, 2020.

[13] 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," in *Proc. Interspeech*, 2019.

[14] X. Mei, C. Meng, H. Liu, Q. Kong, T. Ko, C. Zhao, M. D. Plumbley, Y. Zou, and W. Wang, "WavCaps: A ChatGPT-assisted weakly-labelled audio captioning dataset for audio-language multimodal research," *arXiv preprint arXiv:2303.17395*, 2023.

[15] A. v. d. Oord, Y. Li, and O. Vinyals, "Representation learning with contrastive predictive coding," *arXiv preprint arXiv:1807.03748*, 2018.

[16] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel, "Self-critical sequence training for image captioning," in *Proc. CVPR*, 2017.

[17] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. NeurIPS*, 2014.

[18] R. A. Fisher, "On the mathematical foundations of theoretical statistics," *Philosophical Transactions of the Royal Society of London*, 1922.

[19] H. Robbins and S. Monro, "A stochastic approximation method," *The Annals of Mathematical Statistics*, 1951.

[20] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional Transformers for language understanding," in *Proc. NAACL-HLT*, 2019.

[21] J. F. Gemmeke, D. P. Ellis, D. Freedman, A. Jansen, W. Lawrence, R. C. Moore, M. Plakal, and M. Ritter, "Audio Set: An ontology and human-labeled dataset for audio events," in *Proc. ICASSP*, 2017.

[22] 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*, 2021.

[23] P.-Y. Huang, H. Xu, J. Li, A. Baevski, M. Auli, W. Galuba, F. Metze, and C. Feichtenhofer, "Masked autoencoders that listen," in *Proc. NeurIPS*, 2022.

[24] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark *et al.*, "Learning transferable visual models from natural language supervision," in *Proc. ICML*, 2021.

[25] B. Elizalde, S. Deshmukh, M. Al Ismail, and H. Wang, "CLAP: learning audio concepts from natural language supervision," in *Proc. ICASSP*, 2023.

[26] Y. Wu, K. Chen, T. Zhang, Y. Hui, T. Berg-Kirkpatrick, and S. Dubnov, "Large-scale contrastive language-audio pretraining with feature fusion and keyword-to-caption augmentation," in *Proc. ICASSP*, 2023.

[27] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive text embedding benchmark," *arXiv preprint arXiv:2210.07316*, 2022.

[28] Y. Takida, T. Shibuya, W. Liao, C.-H. Lai, J. Ohmura, T. Uesaka, N. Murata, S. Takahashi, T. Kumakura, and Y. Mitsufuji, "SQ-VAE: Variational bayes on discrete representation with self-annealed stochastic quantization," in *Proc. ICML*, 2022.

[29] Y. LeCun, B. Boser, J. Denker, D. Henderson, W. Hubbard, and L. Jackel, "Handwritten digit recognition with a back-propagation network," in *Proc. NeurIPS*, 1989.

[30] A. Graves, "Sequence transduction with recurrent neural networks," in *Proc. ICML Workshop on Representation Learning*, 2012.

[31] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *Proc. ICLR*, 2019.

[32] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a method for automatic evaluation of machine translation," in *Proc. ACL*, 2002.

[33] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*, 2004.

[34] S. Banerjee and A. Lavie, “METEOR: An automatic metric for MT evaluation with improved correlation with human judgments,” in *Proc. ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, 2005.

[35] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, “CIDEr: Consensus-based image description evaluation,” in *Proc. CVPR*, 2015.

[36] P. Anderson, B. Fernando, M. Johnson, and S. Gould, “SPICE: Semantic propositional image caption evaluation,” in *Proc. ECCV*, 2016.

[37] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft COCO: Common objects in context,” in *Proc. ECCV*, 2014.

[38] S. Liu, Z. Zhu, N. Ye, S. Guadarrama, and K. Murphy, “Improved image captioning via policy gradient optimization of spider,” in *Proc. ICCV*, 2017.

[39] Z. Zhou, Z. Zhang, X. Xu, Z. Xie, M. Wu, and K. Q. Zhu, “Can audio captions be evaluated with image caption metrics?” in *Proc. ICASSP*, 2022.

[40] N. Reimers and I. Gurevych, “Sentence-BERT: Sentence embeddings using siamese bert-networks,” in *Proc. EMNLP*, 2019.

[41] M. T. Ribeiro and S. Lundberg, “Adaptive testing and debugging of NLP models,” in *Proc. ACL*, 2022.