Tokenizing audio: Pipelining TokenLearner into Transformer Models

Alex Young
axyoung@ucdavis.edu
University of California, Davis
Davis, California, USA

Stan He
sbhe@ucdavis.edu
University of California, Davis
Davis, California, USA

Melissa Liu
maliu@ucdavis.edu
University of California, Davis
Davis, California, USA

Fatima Kazi
fikazi@ucdavis.edu
University of California, Davis
Davis, California, USA

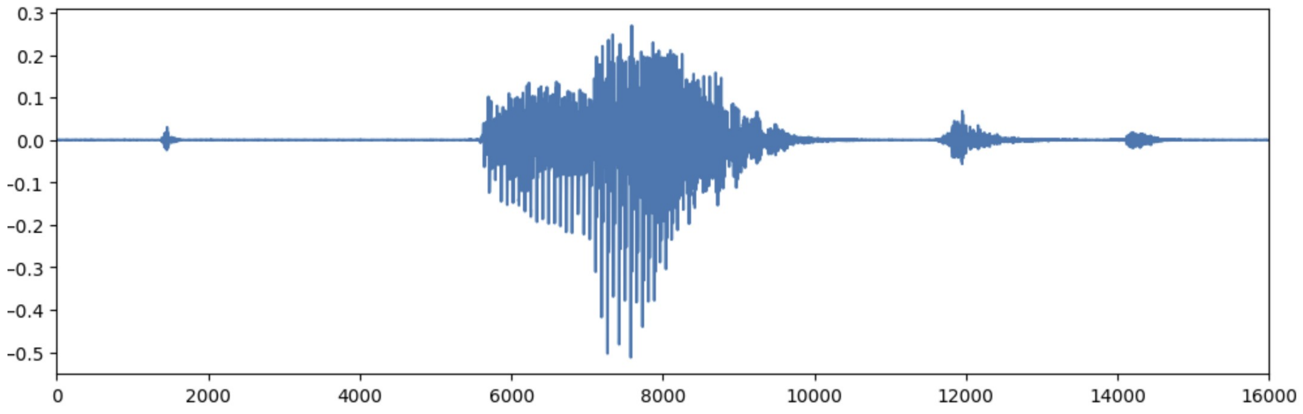

Figure 1: Audio file Waveform visualization [5]

KEYWORDS

TokenLearner, Audio Processing

ACM Reference Format:
Alex Young, Stan He, Melissa Liu, and Fatima Kazi. 2023. Tokenizing audio: Pipelining TokenLearner into Transformer Models. In

Proceedings of (University of California, Davis). ACM, Davis, CA, USA, 9 pages.

1 ABSTRACT

Speed is one of the most important factors for practical, real world, machine learning applications. A model may provide a great result for some task, but if it takes too much time then it is not always practical to be used. A field where this is important is audio applications. If asking Siri or Alexa took 5 seconds to respond to an audio command then the user would feel that they are not very convenient, or for situations where extremely fast response times is necessary, then having an AI respond even faster than before is beneficial. As a result, we seek to increase the speeds of audio recognition tasks by utilizing Token Learner which automatically learns tokens to be passed into a transformer. Overall, Token Learner provides FLOP reduction of up to 10x the FLOPS, but at the cost of accuracy for more complicated datasets.

2 INTRODUCTION

Audio analysis is a prominent and transformative field that combines several areas of research such as Natural Language

Processing (NLP) and human-computer interaction. At the forefront of this domain is automatic speech recognition (ASR), which involves interpreting the spoken language and processing elements such as the language, sentiment, or the words themselves. With the increasing capacity of GPUs, the rise of deep learning, and the unparalleled availability of data, there are more tools than ever to detect and learn patterns that would have otherwise gone unnoticed. Audio processing has continuous effects on several industries and has seen far-reaching implications on low-resource languages and low-literacy populations.

The deep learning regime has eventually outperformed and overtaken older audio analysis methods such as Gaussian mixture models, hidden Markov models, and non-negative matrix factorization (Purwins 2019) [7], As such, audio processing and modeling have often followed the methodology of computer vision. While domain and data differences have pushed unique changes for audio, which compared to images, have chronological order and language constraints, emerging tools such as transformers for NLP (Vaswani 2017) [11] and Vision Transformers ViTs (Dosovitskiy 2020) [2], have challenged the need for permutation sensitive data. Yuan Gong et al. (2021) [3] address this issue with learnable positional embeddings, taken from the architecture of ViTs and with no convolution layers at all.

With the limitations of Moore's Law, a common problem across deep learning is the scaling the size of data. While research continues to show that large models have emerging generalizable and tunable abilities, the billions of parameters necessary to tune these models present a considerable computation challenge. Very few have access to the computational power necessary to train and further scope this domain. Coupled with the natural size difference between audio and video files versus language, there arises a strong need for effective and powerful feature extraction and data compression.

Feature extraction can involve several methods such as principal component analysis, autoencoding, or dimensionality reduction. We will be looking at an adaptive tokenizer developed for images and videos called TokenLearner (Ryoo 2022) [8], and applying it to transformed audio data.

3 RELATED WORK

3.1 Audio Compression

Perhaps the most common application of data compression for audio files is MP3, a lossy audio-data compression algorithm that uses auditory masking to remove or reduce sounds outside the range of normal hearing capabilities. Audio codecs, which simply refer to devices that encode and decode audio, are also used to compress audio for the intention of more accessible storage requirements or network bandwidth, with conceptually no perceptually auditory differences (Zeghidour 2021) [14]. Encoders and decoders are trained to convert high quality audio into the least amount of bits. Compression of audio files in this context are conducted with the intentions of maintaining sound quality and ease of file flexibility. We will be looking at compressing audio files for the intention of training a classifier, which prioritizes different elements of audio, including the zero crossing rate, the rate of sign changes along a signal; spectral rolloff, the frequency below a specified percentage of total spectral energy; and Mel-Frequency Cepstral Coefficients, a description of the overall shape of the spectral envelope (Doshi 2018) [10].

3.2 TokenLearner

Posed with the reality that images have areas of concentrated and diluted semantic strength, and the often hand-designed and impractical splitting strategies to obtain high quality vision tokens, researchers at Google, Michael Ryoo et al. (2022) [8] introduce TokenLearner, which mines important tokens and adaptively learns to find them in spatial content. Their original motivations involved the huge amount of patches for training a model on videos. A video can contain countless frames ranging from 24 to 60 frames per second and to train a model, each frame often needs to be segmented and in their case passed into a transformer. In order to reduce this massive computation time, the authors wanted to allow the model to automatically learn what tokens are the most important per frame, allowing for cases where only 8 tokens needed to be produced per frame which drastically reduces the GLOPs a model takes. Their methods have shown great success for ViTs and video processing, but there is not much data on how effective TokenLearner is on audio. As a result, we seek to apply TokenLearner to Automatic Speech Recognition and other audio tasks. The purpose of this is to see if we can also emulate the success TokenLearner has had in ViTs with keeping accuracy and drastically increasing the speed of our model which is always an important factor.

While still within the domain of ViTs and video processing, TokenLearner has also found success in other applications of machine learning. In particular Anthony Brohan et al. (2022) [1] apply TokenLearner to their 'robotics transformer' to subsample 81 tokens from pre-trained EfficientNet layer to 8 tokens. In doing so, the model is able to greatly improve on its speed of inference and computation, which is a vital aspect of real-world applications. While the direction of our paper strays away from ViTs, the real-world importance of reducing inference time still holds true. In the domain of audio models, this aspect of TokenLearner could be used for tools such as real-time translation or speech recognition.

In tandem with TokenLearner, Google research introduces TokenFuser, which is another module meant to work with TokenLearner. This fuser is meant to remap the TokenLearner representation back to its original spatial resolution [8]. This could potentially help the model make use of and potentially learn from the original spatial representation.

3.3 Spectrogram

Representing audio data to feed through deep learning models is also not obvious and varies depending on the application. Audio features include spectral centroid and high-order statistics of spectral shape, zero crossing statistics, harmonicity, fundamental frequency, and temporal descriptions (Wyse 2017) [13]. Audio convolutional neural networks and transformers generally use spectrograms, 2D images representing sequences of spectra with time along one axis, frequency along the other, and brightness or color representing frequency strength at time t (Wyse 2017) [13]. Yuan Gong et al (2021) [3] process input audio waveform of t seconds, convert it to a 128x100 spectrogram as the input to the AST, split the spectrogram into a sequence of N 16x16 patches, and flatten patches with a linear projection layer that is used as the embedding. An additional trainable positional embedding is added to mimic the spatial structure of the audio spectrogram.

For our application, we take an audio file in the .wav format, convert it into a spectrogram, apply convolution layers on it, and then from the output of those convolution layers we tokenize and then feed those patches into TokenLearner to obtain 8 tokens. As a result, instead of having 64+ tokens, we only have 8 which drastically reduces the training and processing time for our model.

4 METHODS

Dataset Name	Dataset Size	Number of Classes
Mini_Speech_Commands	8,000	8
*ESC*50	2,000	50
*ESC*50*_Condensed*	2,000	5

Table 1: Dataset

4.1 Dataset

We use two different datasets, one with a condensed version, making a total of three. Speech Commands, an audio dataset of spoken words designed with the intention to train keyword spotting systems, for example "Hey Alexa" for Amazon Echo systems or "Hey Siri" for Apple systems (Warden 2018) [12]. The original dataset consisted of 105,829 utterances of 35 words and we use a subset of this dataset, which we refer to as *Mini Speech Commands*, which was created by Tensorflow. This dataset instead contains 8000 audio samples with the classes of $left$, $right$, up, $down$, yes, no, up, and go, and each audio sample is a one second clip of the command being spoken [12].

The second dataset is ESC: Environmental Sound Classification, which is an annotated collection of 2,000 short clips comprising 50 classes of various common sound events (Piczak 2015) [6]. We refer to this dataset as ESC50. The dataset is also further loosely grouped into five major categories with 10 classes per category. We call this version of the dataset with five categories ESC50 Condensed. The author also provides an evaluation of human accuracy to benchmark model performance, which is around 81.5%, with variance in accuracy for certain categories ranging from 34.1% for washing machine noise and almost 100% for crying babies.

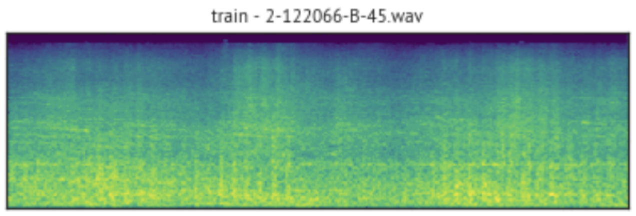

Figure 2: Spectogram of a train [6]

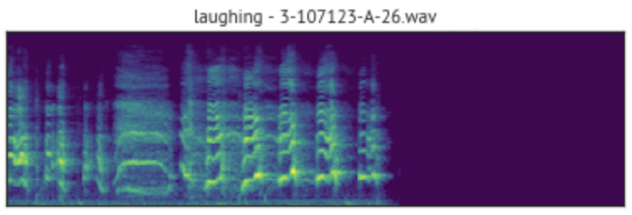

Figure 3: Spectograph of a laugh [6]

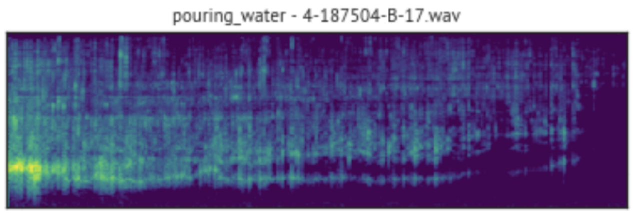

Figure 4: Spectograph of pouring water [6]

4.2 Model Design

For the architecture of our base model, we first take the spectrogram and feed it into a Convolutional Layer then we tokenize its output into 64 tokens. We pass these tokens into TokenLearner which outputs 8 tokens. Positional encoding is applied to these tokens and then put through 8 transformer layers where we feed the output into a dense layer and through a softmax function. The loss function we use is Categorical Cross Entropy Loss. The implementation of these models is through keras.

In addition to this base model, we add MBConv Blocks [9] which includes three layers of 2D convolutions, batch normalization, and activation using ReLU. These blocks are implemented as linear sequential layers.

The ESC50 and ESC50 Condensed datasets are run using the base model design. The Mini Speech Commands dataset is tested using the base model design with an MBConv Block and using two MBConv Blocks.

The TokenLearner implementation that we use is provided through keras as well. This implementation makes use of TokenLearner to demonstrate its improved computation in comparison to a miniViT (Gosthipaty and Paul, 2021) [4].

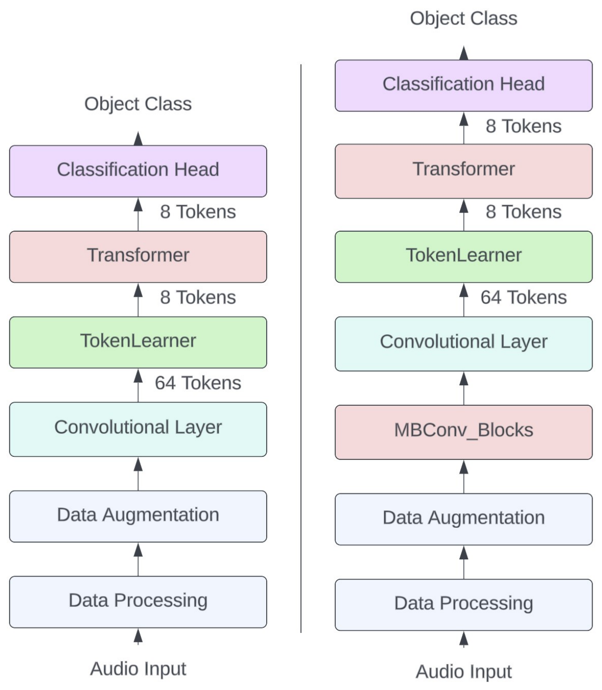

Figure 5: This diagram visualizes our base model (left) and larger models incorporating MBConvBlocks (right)

4.3 Training

For the hyperparameters of our training, we used AdamW for the optimizer to improve the accuracy of the model through use of stochastic gradient descent to improve the decay weights. In addition to AdamW, our model uses keras's learning rate scheduler to apply an updated learning rate to the optimizer. We use both a base learning rate and weight decay of 1e-4.

Our batch size is 64 and we train for 20 epochs at a time in both models. When working with the Mini Speech Commands dataset we resize the inputs to a size of 64x64 and with the ESC50 dataset the inputs are resized to 64x64.

We use TokenLearner to tokenize to 8 tokens and compare to a base model that does not use token learner to tokenize the inputs.

4.4 Performance Evaluation

We measure the computational complexity of training the model with the number of floating point operations per second (FLOPS), with lower FLOPS indicating that the model is running at a lesser cost. The FLOPS count was calculated using keras' built-in counter for 1 batch at a time. By keeping track of the FLOPS used by the model for inference across different hyperparameters and datasets, we are able to evaluate the computational complexity of the model at performing audio tasks with and without token learner.

In addition to measuring the computational complexity of our model, we also measure the accuracy of its classification results. This accuracy is calculated through running the model on the test dataset after training for 20 epochs. While the accuracy is incomparable across different datasets with different types of categories and audio data, it is useful for comparison between running the model with and without TokenLearner.

5 EXPERIMENTS

We follow the problem categorization setup by Purwins et al. (2019) [7], testing our model with an adapted TokenLearner on its sequence classification abilities–predicting a single class label, and its multi-label sequence classification abilities–where the target is a subset of the set of possible classes.

6 RESULTS

For the Mini Speech Commands dataset, we tested several different iterations of our model, using different hyperparameters and layers. Overall, we found that for every model configuration, using TokenLearner significantly reduces the FLOPS of our model ranging from 1.79x to 3.03x less FLOPS. The best iterations of our model using TokenLearner versus not using TokenLearner had accuracies of 90.26% and 89.9%

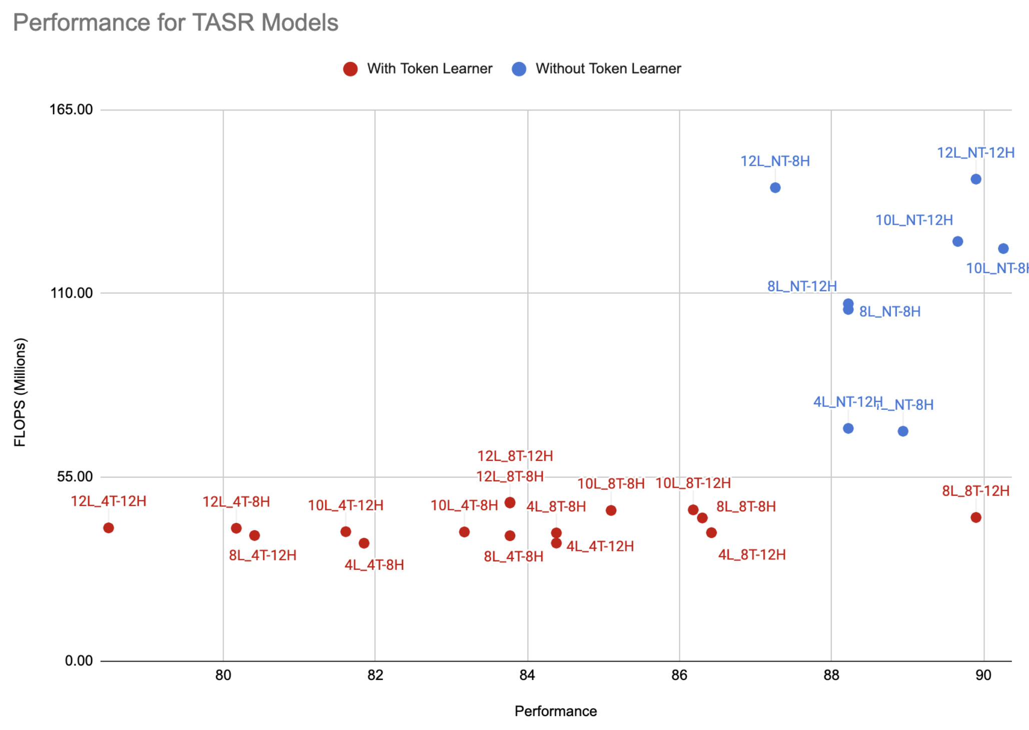

Figure 6: Performance for T-SpeechCommands Models with and without TokenLearner

respectively with the model not using TokenLearner as 2.87x larger in terms of FLOPS.

For the ESC50 and the ESC50 Condensed dataset, we also tested several different iterations of the model, using different hyperparameters and layers. Through all these iterations we found that the models not using TokenLearner were using 7.35x to 10.41x more FLOPS than those with TokenLearner. The difference in computational complexity increased even more in the favor of TokenLearner as the number of layers and complexity of the model increased. In contrast, the best iterations of the model using TokenLearner with the ESC50 dataset has accuracies of 24% and with the ESC50 Condensed the model achieved 39%. Without using TokenLearner the accuracy for ESC50 reached 37.5% and for ESC50 Condensed it reached 50% accuracy.

7 DISCUSSION

Before moving on to discussing the implications of our experimental results, we want to talk about some training issues that arose that heavily alter our conclusions. One major point that was observed in both datasets is an overfitting problem. For the Mini Speech Commands dataset, we observed only a moderate discrepancy in training versus validation data being 98.8% and 87.02% for the non-TokenLearner model and 96.06% and 86.66% for the TokenLearner Model, so while there is a difference in accuracies, it does not heavily sway the results for this dataset.

On the other hand, for the ESC50 datasets, the accuracies have a much larger variance. Some of the highest differences in accuracy is 88.89% training accuracy and 30.77% validation accuracy for the TokenLearner model. As a result, for the results of the ESC50 dataset, it is harder to state conclusions about the efficacy of TokenLearner when considering accuracy tradeoffs without a method of improving the difference in validation and test accuracy. The model may be too complex for our small dataset and as a result, it easily overfits on our data and introducing TokenLearner creates more issues with model parameters to be tuned. However, with the increased complexity of the model, we find that the impact that TokenLearner has on the FLOPS scales quite high to around 10x faster speeds. This difference in computation had quite a visible impact on the amount of time it took to run the classification for these models.

Now for analyzing the model, overall, Token Learner accomplishes its goal of greatly reducing the amount of FLOPS for each of our models. For the simple speech commands dataset, as stated earlier, the highest test accuracy non-TokenLearner model versus highest TokenLearner model only had

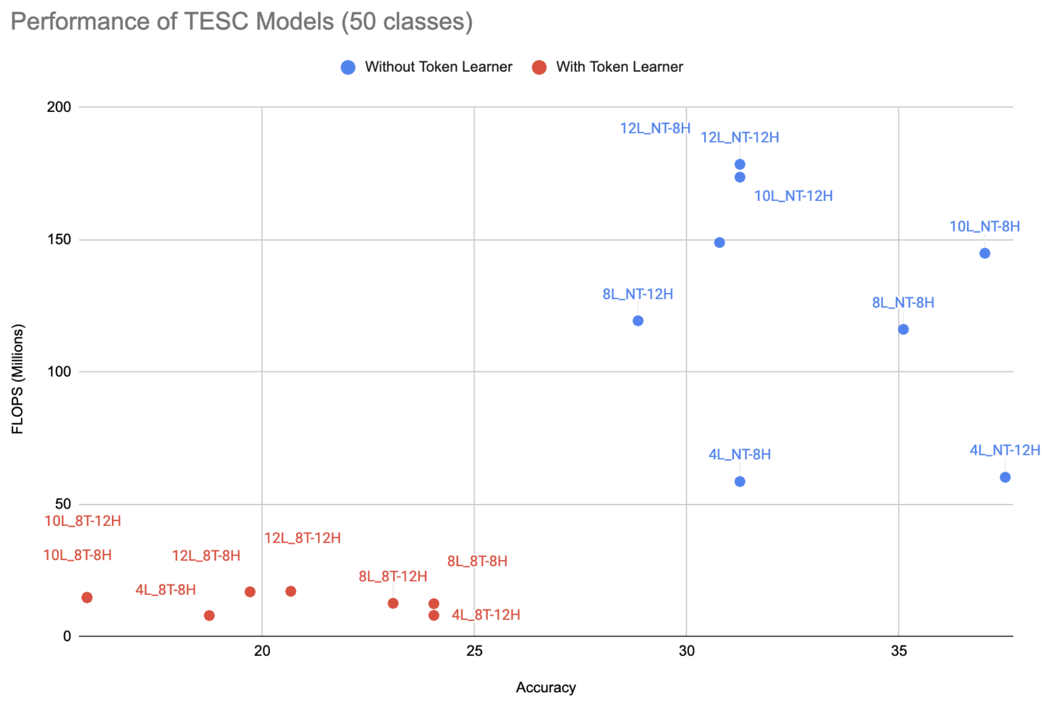

Figure 7: Performance for T-ESC50 Models (50 classes) with and without TokenLearner

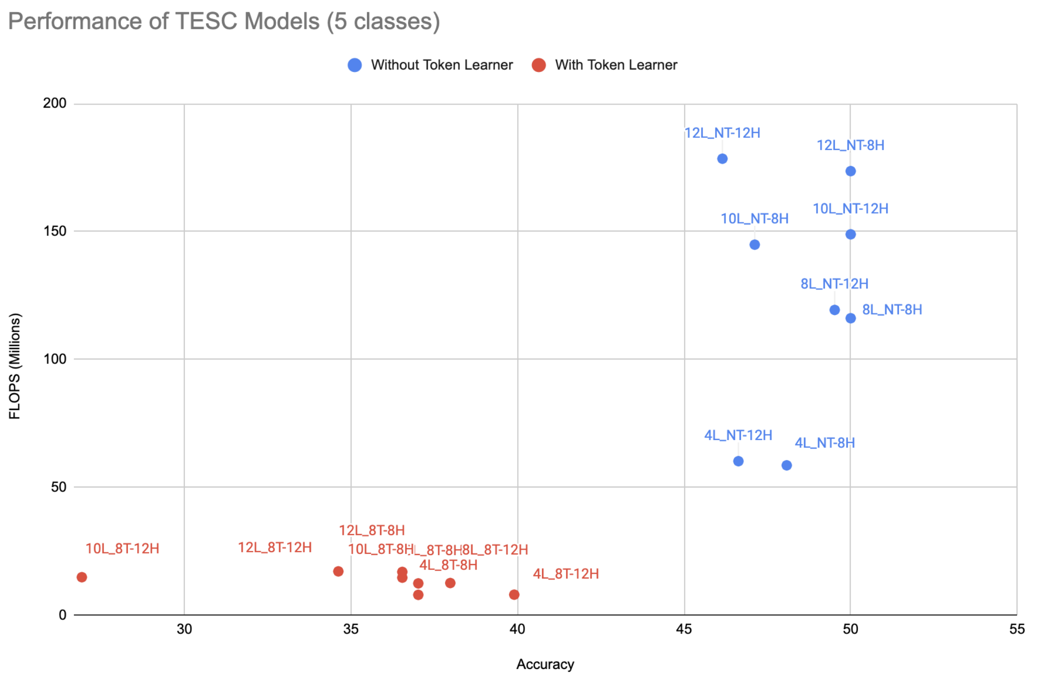

Figure 8: Performance for T-ESC50 Condensed Models with and without TokenLearner

Model	Top-1 (%)	Top-3 (%)	FLOPS
TASR_4L_NT-8H	88.94	97.12	68,792,913
TASR_4L_4T-8H	81.85	96.39	35,295,421
TASR_4L_8T-8H	84.38	96.39	38,369,577
TASR_8L_NT-8H	88.22	97.00	105,232,465
TASR_8L_4T-8H	83.77	96.75	37,534,493
TASR_8L_8T-8H	86.30	97.48	42,852,841
TASR_10L_NT-8H	90.26	97.48	123,452,241
TASR_10L_4T-8H	83.17	96.27	38,654,029
TASR_10L_8T-8H	85.10	96.88	45,094,473
TASR_12L_NT-8H	87.26	97.60	141,672,017
TASR_12L_4T-8H	80.17	95.55	39,773,565
TASR_12L_8T-8H	83.77	96.75	47,336,105
TASR_4L_NT-12H	88.22	97.48	69,644,881
TASR_4L_4T-12H	84.38	96.15	35,329,469
TASR_4L_8T-12H	86.42	96.88	38,440,233
TASR_8L_NT-12H	88.22	97.24	106,936,401
TASR_8L_4T-12H	80.41	95.07	37,602,589
TASR_8L_8T-12H	89.90	97.72	42,994,153
TASR_10L_NT-12H	89.66	97.72	125,582,161
TASR_10L_4T-12H	81.61	96.15	38,739,149
TASR_10L_8T-12H	86.18	96.39	45,271,113
TASR_12L_NT-12H	89.90	97.36	144,227,921
TASR_12L_4T-12H	78.49	95.43	39,875,709
TASR_12L_8T-12H	83.77	96.75	47,548,073

Table 2: Table shows the accuracy of the top one and top three most likely classification results and the respective FLOPS for each model trained on *Mini_Speech_Commands*. The naming convention is as follows: *L* represents number of transformers, *NT* is if TokenLearner is being used with the number being number of tokens, and *H* is number of transformer heads.

a 0.4% difference in accuracy while the TokenLearner model was 2.87x faster. As a result, for this specific dataset, using TokenLearner has extremely minor accuracy reductions while being much faster to process.

Next, looking at the overall accuracies of the ESC50 and ESC50 Condensed datasets, it is understandable why they are so low. While the range of 20% to 35% accuracy for ESC50 and 35% to 50% accuracy for ESC50 Condensed for with and without TokenLearner respectively seems quite low, it is still understandable given the dataset, hardware, and model restrictions. Benchmarks for other very simple models reached similar levels of accuracy without the detriment

Model - 50 classes	Top-1 (%)	Top-5 (%)	FLOPS
TESC_4L_NT-8H	31.25	75.00	58,609,593
TESC_4L_8T-8H	18.75	54.81	7,968,945
TESC_8L_NT-8H	35.10	72.60	116,122,393
TESC_8L_8T-8H	24.04	56.25	12,452,209
TESC_10L_NT-8H	37.02	72.12	144,878,793
TESC_10L_8T-8H	15.87	47.12	14,693,841
TESC_12L_NT-8H	31.25	70.19	173,635,193
TESC_12L_8T-8H	19.71	50.96	16,935,473
TESC_4L_NT-12H	37.50	74.52	60,228,793
TESC_4L_8T-12H	24.04	58.17	8,039,601
TESC_8L_NT-12H	28.85	68.27	119,360,793
TESC_8L_8T-12H	23.08	58.17	12,593,521
TESC_10L_NT-12H	30.77	71.63	148,926,793
TESC_10L_8T-12H	15.87	49.04	14,870,481
TESC_12L_NT-12H	31.25	70.19	178,492,793
TESC_12L_8T-12H	20.67	52.88	17,147,441

Table 3: Table shows the accuracy of the top one and topthree most likely classification results and the respective FLOPS for each model trained on *ESC50s*.

of needing to condense these large .wav files into a 64x64 matrix. This extreme simplification of the input was due to having to save space in Google Colab. In addition, the dataset itself is far more difficult to perform classification on due to it naturally having many different classes and far fewer data points for the model to learn from. Furthermore, the ESC50 dataset is composed of environmental sounds, which are far more abstract than that of ASR datasets where the model can learn specific speech or language patterns instead of general noises. In specific, losing recognizable language patterns is likely a large reason for the greater difference in accuracy of TokenLearner versus without TokenLearner on the ESC50 datasets than the Mini Speech Commands dataset. TokenLearner attempts to extract more important features from the input and select those tokens to be used by the rest of the model. This may be detrimental to input data that has less obvious patterns or easily recognizable features.

8 CONCLUSION

8.1 Limitations

As mentioned before, one primary limitation is the access to the physical resources: both GPU and computing capacity greatly reduced the potential size of the model. This further hindered the experimental capacity and proposed ideas had to be substantially modified to be fit within the bandwidth of our allotted time and resources.

Another significant limitation to the experimental setup and results is the size of existing datasets and model compatibility. Give the substantial size difference between audio files and text files (with audio files being significantly larger), even a dataset of mediocre sample size is remarkably larger than a text file dataset of a similar sample size. With the already limited GPU, CPU, and RAM capacities, a decently sized dataset still posed significant challenges when downloading and running.

As a result of these hardware limitations, the datasets we used were incredibly small and all the spectrograms we obtained were downsized to 64 x 64. As a result, for audio samples from the $ESC50$ dataset, we greatly reduce the image from a 624 x129 sized input spectrogram to a 64 x 64 sized input which causes massive information loss. Furthermore, this means we did not utilize the full *Speech_Commands* dataset that the *Mini_Speech_Commands* dataset was sampled from. This stopped also us from using more complete sound recognition datasets such as AudioSet.

8.2 Ethical Considerations

Alongside all applications of machine learning, the use of TokenLearner with transformer models and audio input has its ethical considerations to be aware of. TokenLearner selectively narrows down the input tokens to the tokens it feels are the most important. Alongside with the base model itself, TokenLearner is also a layer of the model that can incorporate its own learned biases.

Another consideration for further training of audio models is the importance of protecting the privacy of sampled speakers. While anonymity can easily be maintain within text files, audio files inherently preserve some identity of the speaker with their vocal fingerprint.

8.3 Future Work

One future work that this work gave inspiration to is to insert Token Learner into a SOTA model. What we did was create our own basic Transformer model to take in audio spectrograms, but what we would like to investigate is for some of the pretrained models that have extremely high accuracies on difficult datasets, whether or not TokenLearner can maintain those accuracies while being faster.

Another idea to take into consideration is to train a TokenLearner from scratch on audio spectrograms and compare the similarities in the weights it learns to those from TokenLearner that is based on ViTs.

8.4 Takeaways

What we discovered is that TokenLearner will always lower the FLOPS of any of our models. However, there are several issues that might arise from doing so. There are more parameters to be tuned which makes the model more easily overtune on data and generalize less, and the reduction of Tokens can result in information loss which reduces accuracy.

For real-world applications and for research purposes, having methods of decreasing computational complexity for potentially low downsides can be quite useful. Our findings show that if implemented carefully, TokenLearner can have applications beyond the scope of ViTs. However, this entails understanding and applying TokenLearner to models and datasets that it will perform well with. Hopefully in the future, tools like TokenLearner will help make Transformer models more accessible and approachable for casual use and research.

9 CODE

Following is the link to the Github where our project and experiments reside: https://github.com/StanBHe/Token-Learner-and-Audio-Recognition-Tasks/tree/main.

REFERENCES

[1] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Joseph Dabis, Chelsea Finn, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Tomas Jackson, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Kuang-Huei Lee, Sergey Levine, Yao Lu, Utsav Malla, Deeksha Manjunath, Igor Mordatch, Ofir Nachum, Carolina Parada, Jodilyn Peralta, Emily Perez, Karl Pertsch, Jornell Quiambao, Kanishka Rao, Michael Ryoo, Grecia Salazar, Pannag Sanketi, Kevin Sayed, Jaspiar Singh, Sumedh Sontakke, Austin Stone, Clayton Tan, Huong Tran, Vincent Vanhoucke, Steve Vega, Quan Vuong, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Tianhe Yu, and Brianna Zitkovich. 2022. RT-1: Robotics Transformer for Real-World Control at Scale. arXiv:2212.06817 [cs.RO]

[2] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. 2021. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. arXiv:2010.11929 [cs.CV]

[3] Yuan Gong, Yu-An Chung, and James Glass. 2021. Ast: Audio spectrogram transformer. *arXiv preprint arXiv:2104.01778* (2021).

[4] Aritra Roy Gosthipaty and Sayak Paul. 2021. Learning to tokenize in Vision Transformers. https://keras.io/examples/vision/token_learner

[5] Tarun Jain. 2020. Simple audio recognition: Recognizing keywords. https://www.tensorflow.org/tutorials/audio/simple_audio

[6] Karol J. Piczak. [n. d.]. ESC: Dataset for Environmental Sound Classification. , 1015–1018 pages. https://doi.org/10.1145/2733373.2806390

[7] Hendrik Purwins, Bo Li, Tuomas Virtanen, Jan Schlüter, Shuo-Yiin Chang, and Tara Sainath. 2019. Deep learning for audio signal processing. *IEEE Journal of Selected Topics in Signal Processing* 13, 2 (2019), 206–219.

[8] Michael S Ryoo, AJ Piergiovanni, Anurag Arnab, Mostafa Dehghani, and Anelia Angelova. 2021. Tokenlearner: What can 8 learned tokens

do for images and videos? *arXiv preprint arXiv:2106.11297* (2021).

[9] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2019. MobileNetV2: Inverted Residuals and Linear Bottlenecks. arXiv:1801.04381 [cs.CV]

[10] Doshi Sanket. 2018. Music Feature Extraction in Python. https://towardsdatascience.com/extract-features-of-music-75a3f9bc265d

[11] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. arXiv:1706.03762 [cs.CL]

[12] Pete Warden. 2018. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209* (2018).

[13] Lonce Wyse. 2017. Audio spectrogram representations for processing with convolutional neural networks. *arXiv preprint arXiv:1706.09559* (2017).

[14] Neil Zeghidour, Alejandro Luebs, Ahmed Omran, Jan Skoglund, and Marco Tagliasacchi. 2021. Soundstream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 30 (2021), 495–507.