Audio Noise Elimination Using A Transformer Based Recurrent Neural Network

Landon Burleson
Electrical and Computer Engineering
Oklahoma State University
Stillwater, Ok, USA
landon.burleson@okstate.edu

Madhusti Dhasaradhan
Electrical and Computer Engineering
Oklahoma State University
Stillwater, Ok, USA
madhusti.dhasaradhan@okstate.edu

Alex Sensintaffar
Electrical and Computer Engineering.
Oklahoma State University
Stillwater, Ok, USA
alex.sensintaffar@okstate.edu

1. Abstract

Denoising an audio track without altering the integrity of voice data is a complicated task within the field of digital signal processing. Common implementations within the industry include digital filtering using techniques such as wiener filtering or other non-neural network strategies. Some neural network implementations include LSTM (Long Short Term Memory) or the simplified equivalents, GRU (Gated Recurrent Unit). However, these implementations, while do help compute relationships between the time samples, do so with limited success with long time-series sequences. To avoid the limitations of LSTM-based neural network implementations, we are proposing a transformer-based RNN model to be used in a random and environmental audio denoising model. The transformer-based model also has the added characteristic of the ability to be parallelized to maximize throughput and speed to the final output of the neural network model. This speed would allow for real-time audio to be inputted into the trained model and output filtered data with a minimal delay between the input and output. Utilizing both the transformer and RNN-based architectures, we will be able to show the empirical improvements when utilizing the transformer in comparison to the RNN architecture.

2. Introduction

Noise in audio is detrimental to a listener in terms of understanding and enjoyment. Filtering the undesirable frequency ranges in real-time has proven to be computationally heavy in terms of processing power required to achieve this goal. Graphics card manufacturer, Nvidia, has implemented such a feature for their former generation of Graphics Processing Units (GPU) and beyond by leveraging the bandwidth available in the aforementioned processors. However, using the real-time filtering available on this hardware requires immense power and proprietary algorithms. The goal of this project is to propose an open-source Neural Network architecture that is efficient enough to satisfy the real-time filtering offered elsewhere on various hardware. This problem is interesting due to the complexity of accounting for the speed of the neural network and the approach in which to tackle voice recognition within audio tracks or audio channel(s).

Utilizing RNN and transformer-based architectures, we will be able to filter environmental audio and background noise in real time for better clarity to a listener in a virtual meeting/conference setting. The primary datasets that will be used to accomplish this will be the MS-SNSD[1] and our random noise-induced audio tracks with single words being spoken. Our own constructed dataset will be discussed further in later sections of this report. Existing implementations utilized LSTM[2]-based RNNs. We propose utilizing a transformer-based RNN structure to modernize the architecture and allow our model to more easily adapt to long time-series sequences without hindering performance.

3. Background

A recent paper utilized an LSTM-based RNN architecture to provide recognition of voices and similarity between samples[2]. Other implementations, such as Nvidia's, are proprietary and thus encompassed in an unreachable 'black box'. Utilizing these references, we will innovate with an open-source, low-overhead transformer-based RNN model.

Other aforementioned work in this digital signal processing area includes traditional non-ML-based solutions. Among these techniques include the wiener filtering methodology. However, this solution has the drawback of over-filtering the signal and thus degrades the integrity of the desired audio data.

4. Approach

As mentioned before, our proposal to tackle the stated problem is to implement a transformer-based RNN in order to better avoid the vanishing gradient problem as well as take advantage of the throughput benefits the Attention module has to offer within this type of architecture. These

[1] https://github.com/microsoft/MS-SNSD

[2] Z. Li and P. Song, "Audio similarity detection algorithm based on Siamese LSTM network," 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP), 2021, pp. 182-186, doi: 10.1109/ICSP51882.2021.9408942.

throughput benefits would allow us to input a live audio stream into the model and process the filtering of both background noise and environmental noise from the audio stream for a better experience on the listener's side. This technology is both useful and highly desired in conferences, phone calls, and pertinent communications that have extreme background audio levels such as in industrial environments.

The attention module within the transformer would focus on our desired data and thus reduce the overall interference noise would have on the input audio source.

The first set of data that was used stems from the MS-SNSD dataset. This dataset has both clean and noisy voice audio files that can be used to train our proposed neural network architecture to filter noise and focus on speech patterns. Another dataset that may be used is the audio MNIST dataset which has short audio recordings of low-noise voices reading simple sentences. This could be used in conjunction with the MS-SNSD dataset to better recognize voices within our network architecture. See the footnotes for hyperlinks to both of the described datasets. This data was used while going through initial testing and creation of the RNN.

The data was later changed to the TensorFlow mini_speech_commands data set. This data is small one-second sounds that are easy to mass produce as needed. The mini_speech_commands consist of only clean sounds with little to no noise. The advantage of the mini_speech_commands is multiple different voices being used for each command word which gives us a lot of room to choose the best combination of different words and sounds per word.

The random noise was created based on a normal Gaussian distribution. The root mean square of the signal is used to determine the scale of the distribution and the shape of the signal is used to determine the size of the distribution. When played the voice saying the word can still be heard but there is clear and noticeable static in the background.

$$RMS = \sqrt{mean(signal^2)}$$

Equation 1. RMS Equation used to find the RMS of the sound file

The following graphs are used to showcase a signal with and without noise for the same audio track. Notice the noise encompasses the lower amplitude details and leaves the relatively high amplitude portion of the audio relatively untouched. By observation, the original signal is difficult to discern without viewing the original audio track spectrum. In this use case, it would be difficult to utilize standard DSP filtering techniques as the noise is random and populates the entire frequency spectrum. The attention module inside our proposed model will yield excellent results recognizing voice data versus noise as those features will be the most prevalent in the model when compared against the clean audio versus the noisy audio. The process by which we will be evaluating performance will be discussed in the next section of the report.

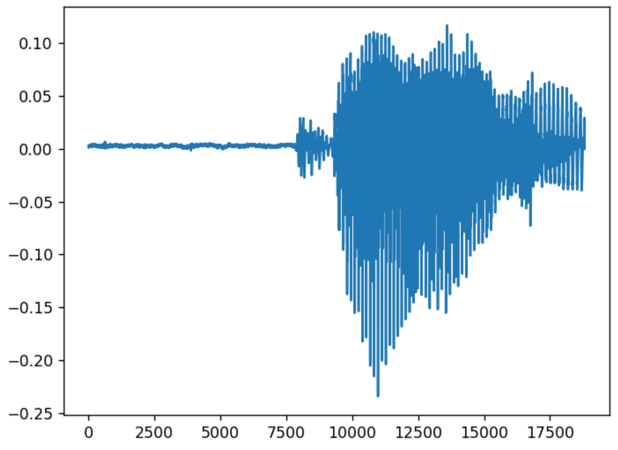


Figure 1. Initial Sound Without Noise

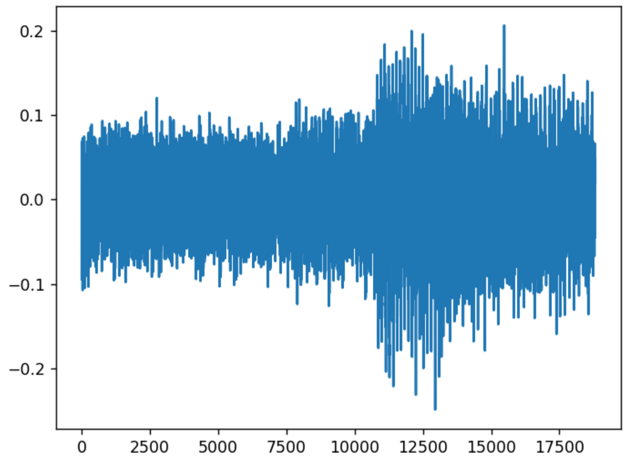


Figure 2. Sound File with Random Noise Added

Sound File Without Noise
Sound File With Noise

In addition to the manual audio input vs output review, among the metrics that will be used to distinguish the difference between the clean audio sample and the environmental/random noise input sample will be the mean squared error (MSE). This metric is often used to find the similarities between two sets of regressive data. With the datasets being a regressive dataset rather than a categorical type, this metric will easily find the closeness between the actual and the predicted sequential data samples.

5. Code Reuse:

The vast majority of our code is original work. However, much of the code has outside influences and thus, must be properly cited. The 'PositionalEncoding.py' file used within the transformer-based model utilized code found in a GitHub repo [13]. Other references used in aid of attempting to create the transformer model stem from the Keras documentation examples for Natural Language Processing based transformer models [10][11][12][14].

The code to model, simulate and add noise to the initial sound file is based on Wijayasingha's [2] article on how to add noise to a sound file. Wijayasingha provided the noise equation as well as the code to plot the waveforms of the sound files. Another code that was used within the repository is directly based upon the documentation for the Librosa library used to build our train, test, and validation data. Otherwise, unless denoted inside of comments within the code itself, the structure and organization of the code is original work.

The main.py, graph.py, and portions of the model.py and data.py scripts were original work. Functions or files that utilized outside influences were accounted for and a URL pointing to the original work was provided in the comments of the function or file.

6. Experiment:

This section provides a background in the experimentation done using the noisy and non-noisy audio tracks for both the RNN and Transformer based model. The subsequent subsections discuss the data and the strategies used for the Neural Network models that we used and compared against. However, due to time constraints, the transformer-based model failed to materialize in a working state. All code that was being used to build the transformer, however, is still present in the code repository and the evaluation process that would have been used to test this model is included in this report in order to provide the semantics to accompany this transformer model in the event that this project continues.

6.1 Data:

The noisy data was created from the mini_speech_commands files. The mini_speech_commands consist of 8 words down, up, go, stop, left, right, no, and yes. Each word is spoken 1000 times by different voices and each word is stored as separate .wav files. For each sound, noisy copies can be created by adding in random noise using the RMS equation. Due to size concerns, we created only 100 noisy files randomly selected from the 8 words. Each noisy file was created and stored along with its respective clean copies. The 100 noisy files are randomly made up of the 8 words and a random subset of sounds from the 8 words. The 100 files were randomly separated into 60 training files, 20 testing files, and 20 verification files. The encompassing .zip file is a total of 4.37 MB.

6.2 RNN Evaluation

The primary source of data is described in the previous section of this report as various speakers saying simple one-word responses. However, these audio tracks were clean when they were obtained. We used an RMS noise model to introduce random white noise into the audio tracks in order for us to remove it with our proposed models. Using this synthesized data, we were able to split this data set into training, testing, and validation data sets. Once trained and evaluated using our testing data, we inputted 10 noisy audio tracks into the model and the predictions were compared against the input. Our proposed model was evaluated using the audible quality of the predicted outputs. The model itself was verified using the Mean Square Error values determined after every epoch for 25 epochs.

The following graph showcases the initial MSE versus epoch graph for about 30 training samples for the finalized RNN model as shown in *Figure 4*.

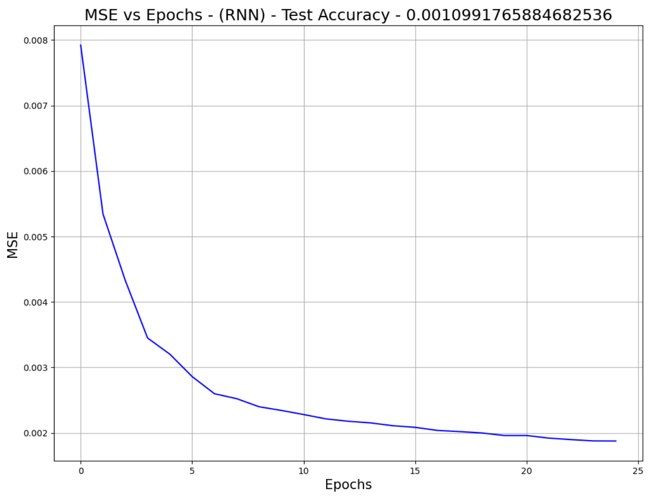


Figure 3. MSE vs Epochs for the RNN model

The above result represents the MSE values from inputting an audio track with approximately 22048-time steps within the track. These raw audio files were inputted directly into the RNN to achieve this type of response. See Figure 7 to view the Finalized RNN structure used to achieve the above response.

As shown in Figure 7, the finalized structure of the baseline RNN is shown with one SimpleRNN and a deep learning network post RNN layer. This structure, as well as the number of neurons/cells in each, were chosen based upon the best performing prediction outputs. Some samples of these outputs are shown below as both audio tracks and the accompanying waveforms. The waveforms can be viewed with figures 4 through 6.

The following audio sample input was placed into the RNN model with the output being denoted as 'predicted'. Notice the output audio track does reduce the majority of the noise. However, the overlap of the noise with the vocal frequency range seems to introduce some level of distortion. The output audio track also has a noticeably lower audio level than the input. The RNN seems to over-adjust for the noise by reducing the audio but also does reduce some of the noise for some of the input tracks. One more thing of note is the low amplitude vocal lobe on the left side of the ‘Actual’ waveform shown in *Figure 5*. When the noise was introduced to the input waveform, shown in *Figure 4*, the noise floor fully encompasses this lobe of the vocal range. This implies the underlying vocal data that has an amplitude less than the noise floor will be much more difficult to reproduce in traditional DSP filtering means. In this circumstance, the RNN model would be more versatile in the ability to interpolate the lost data and reproduce these seemingly destroyed data. However, given the results we have shown, improving the model would be imperative in providing this functionality. More advanced implementations such as Transformers would also provide a benefit in recreating some of these vocal tracks.

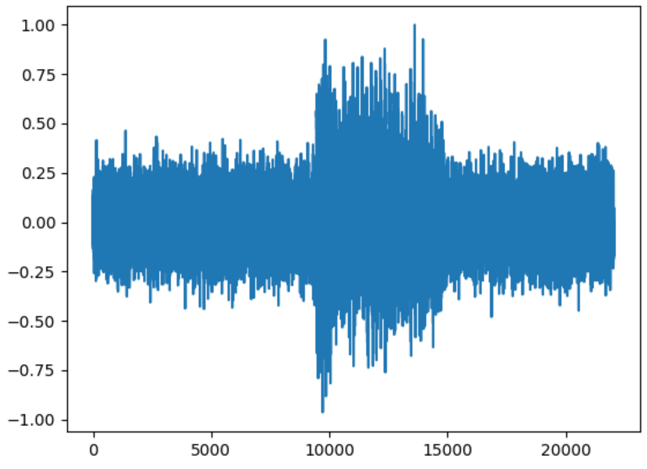


Figure 4. Noisy Input Audio Waveform Plot

Input Noisy Audio File

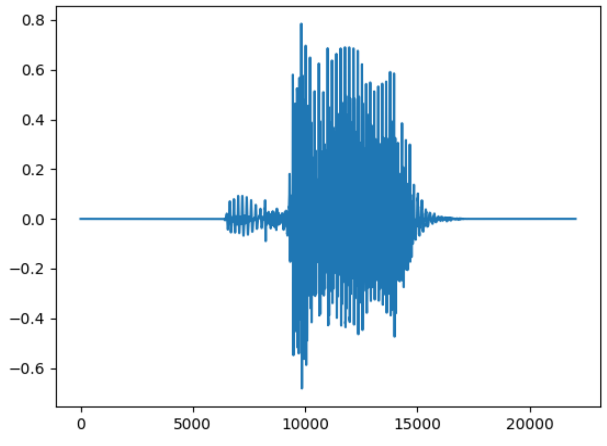


Figure 5. Actual Audio Waveform Plot

Sound File Without Noise

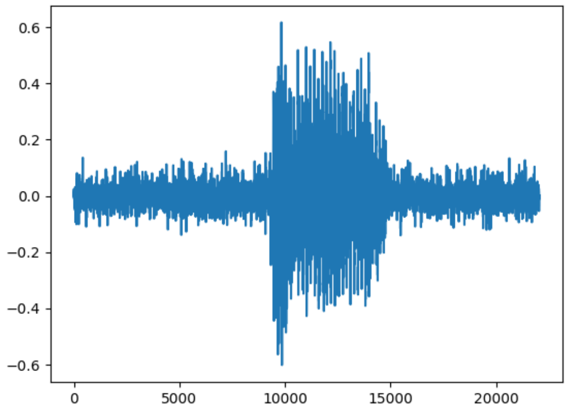
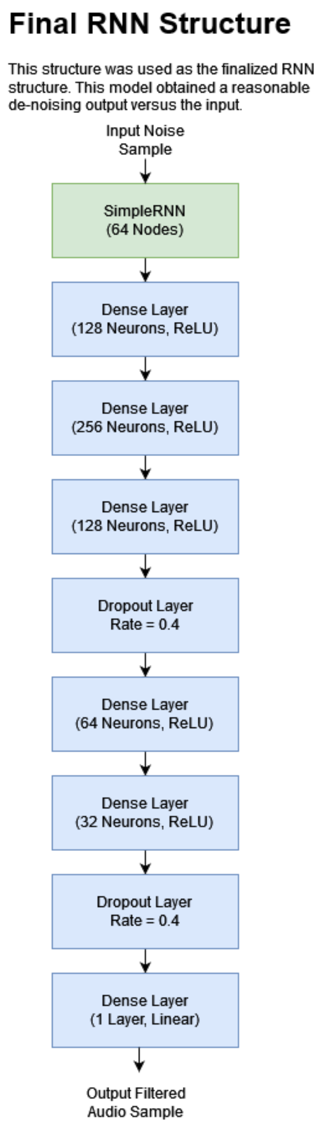


Figure 6. Predicted Sound Waveform Plot

Predicted Sound File



Final RNN Structure

This structure was used as the finalized RNN structure. This model obtained a reasonable de-noising output versus the input.

Input Noise Sample
SimpleRNN (64 Nodes)
Dense Layer (128 Neurons, ReLU)
Dense Layer (256 Neurons, ReLU)
Dense Layer (128 Neurons, ReLU)
Dropout Layer Rate = 0.4
Dense Layer (64 Neurons, ReLU)
Dense Layer (32 Neurons, ReLU)
Dropout Layer Rate = 0.4
Dense Layer (1 Layer, Linear)
Output Filtered Audio Sample

Figure 7. Finalized RNN model.

The above structure is used to provide a baseline for our project by utilizing an existing architecture used in DSP applications for comparison against the state-of-the-art transformer-based design that we are proposing. Unfortunately, the transformer based model was not functional at the time of writing this report. For prospective future work, we would implement the model shown in *Figure 8* with the exception of the “Softmax” function shown on the output of the diagram. This figure was taken from a Natural Language Processing (NLP) transformer models where classification was more prevalent than the regressive based operations that we propose in our work shown in this report.

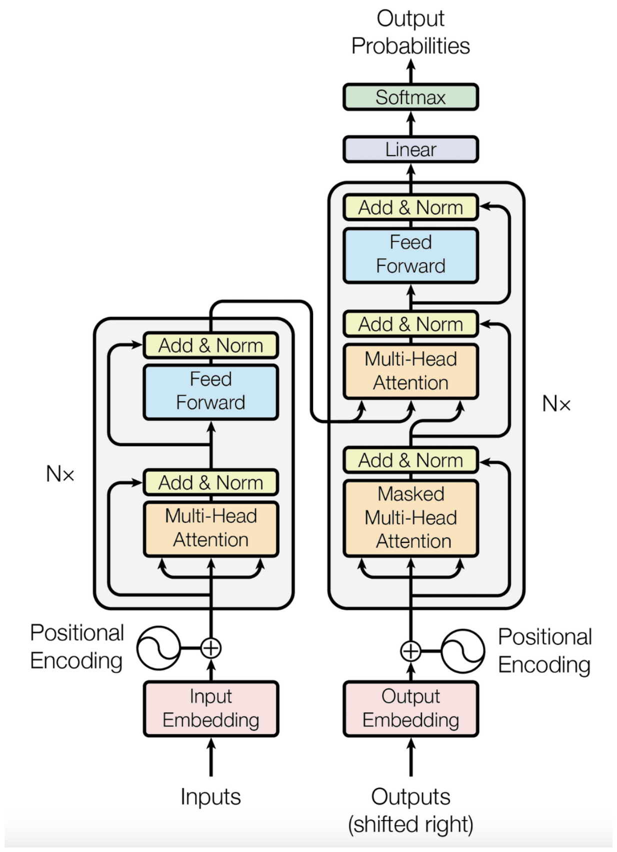


Figure 8. Transformer model diagram courtesy of “Attention is all you need” paper [12]

6.3 Transformer Evaluation

The proposed transformer utilized the structure shown above and was implemented with output modifications. The ‘softmax’ output at the top of the decoder was removed in favor of outputting the desired continuous regressive output. The layer prior to this utilized a linear output to ensure the continuous characteristic. The implementation of the transformer was done by incrementally creating each portion of the above network using the Keras functions. As mentioned in the ‘Code Reuse’ section above, the transformer used in our code was heavily influenced by several works found online. However, the majority of the references were strictly classification based rather than regression. Due to this, we modified the transformer structure to be regressive-based rather than a classification-based model. However, with these modifications to the implementation, we were unable to get the transformer portion of the project to run appropriately within the timeframe. Future work prospects are discussed in the ‘Conclusion’ section of this report.

7. Conclusion:

The usage of RNN and transformer-based neural networks are exceptionally useful in providing a framework for dynamically removing noise within input audio tracks. Due to the time constraints we were seeing, we opted to only use one feature (the audio sequence) for the feature engineering portion of the project with the optimism that adjusting the hyperparameters of the networks yields the greatest improvement in performance. This unfortunately proved to be inaccurate. In continuation of this project, we would add more features on top of the audio track to improve the network's response to the data. One such feature set would include the FFT output from each of the noisy and non-noisy audio tracks to improve the visualization of the noise data in the frequency domain. This would greatly improve the network's response to the noise and greatly improve the performance. Another improvement would be to finely tune the hyperparameters once more features are included in the dataset. This would result in greatly improved accuracy and MSE values for both training and testing datasets. Using the spectrum instead of the raw audio data would benefit the performance of all denoising models tremendously. Especially if used in tandem with convolutional-based layers to extract the exact moments of voice data. This would also help the attention mechanism within the transformer model to focus on the vocal classification.

This project provided a lot of new knowledge not previously known by any of us. Using the Hugging Face website for transformer-based models rather than building a transformer from scratch seems to be highly useful for tasks requiring large time sequences to obtain a prediction for such data types. Given the trouble we had creating a transformer for denoising an audio input, utilizing the Hugging Face resource would be highly beneficial for a future implementation of this project. Also learned that feature extraction and use in neural networks is very important and requires an extended period of time to perform appropriately in a real-world application. The simple RNN model used as our baseline model performed well once the network was further tuned following the initial design. However, the recreation of the smaller amplitude vocal data lacked interpolation or complete elimination of the noise floor as shown in *Figure 6*. Overall, these networks provide a unique solution to denoising a highly dynamic and random dataset.

8. Division of Labor

The following table represents our division of labor

Task	Landon	Madhusti	Alex
Plan Project Idea	X	X	X
Write on Project Proposal	X	X	
Create Train and Test Data			X
Create RNN	X		
Create Transformer	X		
Write Midterm Report	X	X	X
Test RNN	X	X	
Test Transformer	X		
Write Final Report	X	X	X

To date, we as a group have collected all data samples we will use to train and test our model. We have created many data points to split the testing and training data with the relative number of points referenced in the approach section above. The RNN portion of the project was completed with extensive testing and evaluation. The transformer-based model was partially completed due to timing constraints. If we were to continue the project, we would use a built transformer-based model using the Hugging Face database and modify the model as needed.

9. References

[1] C. K. A. Reddy, E. Beyrami, J. Pool, R. Cutler, S. Srinivasan, and J. Gehrke, ‘A Scalable Noisy Speech Dataset and Online Subjective Test Framework’, Proc. Interspeech 2019, pp. 1816–1820, 2019.

[2] L. N. Wijayasingha, ‘Adding Noise to Audio Clips’, Medium, Feb. 2020.

[3] C. R. Harris et al., ‘Array programming with NumPy’, Nature, vol. 585, no. 7825, pp. 357–362, Sep. 2020.

[4] P. Virtanen et al., ‘SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python’, Nature Methods, vol. 17, pp. 261–272, 2020.

[5] J. D. Hunter, ‘Matplotlib: A 2D graphics environment’, Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007.

[6] B. McFee et al., librosa: v0.4.0. Zenodo, 2015.

[7] Z. Li and P. Song, ‘Audio similarity detection algorithm based on Siamese LSTM network’, in 2021 6th International Conference on Intelligent Computing and Signal Processing (ICSP), 2021, pp. 182–186.

[8] Mart’in Abadi et al., ‘TensorFlow Large-Scale Machine Learning on Heterogeneous Systems’. 2015.

[9] Mark Daoust et al., ‘TensorFlow Simple audio recognition Recognizing keywords’. .

[10] T. Ntakouris, ‘Keras Documentation: Timeseries classification with a Transformer model’, Keras. Jun-2021.

[11] C. Stefania, ‘Implementing the Transformer Encoded from Scratch in TensorFlow and Keras’, Machine Learning Mastery. Nov-2022.

[12] A. Vaswani et al., ‘Attention Is All You Need’, CoRR, vol. abs/1706.03762, 2017.

[13] Cdezapasquale, ‘Cdezapasquale/transfomer-audio-classification: Small experimentation about positional encoding’, GitHub.

[14] K. Team, ‘Keras Documentation: Timeseries classification with a Transformer model’, Keras.

[15] Mkvenkit, ‘simple_audio_train_numpy.ipynb’. Jan-2021.