Exploring Deep Learning Techniques for Voice Activity Detection

Name
Department of Computer Science and Engineering
Lovely Professional University
Jalandhar, India
nameexample@gmail.com

Name
Department of Computer Science and Engineering
Lovely Professional University
Jalandhar, India
nameexample@gmail.com

Name
Department of Computer Science and Engineering
Lovely Professional University
Jalandhar, India
nameexample@gmail.com

***Abstract*— Voice Activity Detection (VAD) is a key part of any speech processing system—it helps figure out when someone is speaking and when there's just silence in an audio signal. In the past, VAD systems were based on manually crafted features and rule-based logic, which often didn't work well in noisy or unpredictable environments. But now, with deep learning, those issues are being tackled more effectively. In this study, we explore how deep learning—specifically convolutional neural networks (CNNs)—can improve VAD performance. We trained our model on a wide range of audio samples, including both clean recordings and noisy ones, to make sure it performs well in different situations. The process involved several key steps: going through audio files in a structured way, turning them into spectrograms, and applying data augmentation techniques like adding noise and changing pitch. We fine-tuned our CNN model using techniques like regularization, adjusting the learning rate on the fly, and stopping training early to avoid overfitting. The results show that our approach significantly outperforms traditional methods, especially in terms of accuracy and handling background noise. This supports the idea that deep learning is well-suited for real-time voice-based applications like smart assistants, voice calls, and other speech-driven systems.**

Keywords—Voice Activity Detection, Deep Learning, Convolutional Neural Network, Speech Recognition, Audio Classification, Spectrogram, Noise Robustness

I. INTRODUCTION

Voice A Voice Activity Detection (VAD) is central to modern speech systems, helping figure out whether an audio clip contains actual speech or just noise or silence [1]. This function is essential in everyday tools—telecom systems, virtual assistants, hearing aids, and speech recognition technologies all rely on it [2]. By focusing only on meaningful speech input, VAD cuts down processing needs, saves data usage, and generally makes systems more responsive—especially when low-latency behaviour is expected [1][2].

Back in the earlier stages, most VAD systems used straightforward signal processing tricks. They'd look at basic features like short-term energy, zero-crossing rates (ZCR), or spectral entropy [3]. These methods worked well enough under ideal conditions but quickly ran into problems when real-world factors kicked in—like background noise, multiple speakers, or inconsistent recording setups [3][4]. As systems became more complex, the cracks in these approaches started to show, pushing researchers to look for smarter, more resilient solutions.

That shift led to the adoption of machine learning, and eventually deep learning. Unlike older methods, deep models—such as convolutional neural networks (CNNs) and recurrent neural networks (RNNs)—don't depend on hand-crafted features [5][6]. Instead, they learn patterns from raw audio or spectrograms. With enough diverse data, these models hold up well across a wide range of situations. Inputs like spectrograms or mel-frequency cepstral coefficients (MFCCs) help the models understand both time and frequency information, which is key for accurate detection [7].

In our work, we developed a VAD system based on CNNs. The approach includes preparing data, converting it into spectrograms, and using augmentation steps—such as pitch shifting or adding background noise. The dataset was intentionally varied to make the model flexible. Our experiments showed that the CNN setup not only outperformed older techniques in noisy environments but was also more consistent overall. To help with real-time use, we applied tuning techniques like early stopping, regularization, and dynamic learning rate control. These steps made the system both reliable and lightweight enough for practical use. The findings suggest useful directions for future work in this space [8][9].

II. LITERATURE REVIEW

In the early days, Voice Activity Detection (VAD) systems were mostly built on classic signal processing techniques. These systems relied on manually crafted rules and features, often using pretty straightforward logic. One of the simplest and most widely used methods was energy-based detection. Basically, the idea was to check whether the short-term energy of the audio signal passed a certain threshold—if it did, the system assumed speech was present. While that approach was easy to implement and ran fast, it wasn't very reliable in noisy situations. Background sounds or sudden non-speech noises could easily throw it off, causing either missed speech or false triggers. As time went on, researchers came up with more refined techniques like zero-crossing rate (ZCR), spectral subtraction, and statistical modeling. Standards like ITU-T G.729 Annex B and G.723.1 brought in more advanced decision-making rules and noise suppression techniques [1][2]. Even so, these systems didn't hold up well in unpredictable or noisy environments. They often needed manual tuning to work properly in each new setting.

Because of those limitations, people started turning to machine learning as a better alternative. The first wave of machine learning-based VAD models made use of algorithms like Support Vector Machines (SVMs), Gaussian Mixture Models (GMMs), and Hidden Markov Models (HMMs). These brought in a big advantage: instead of hardcoded rules, the models could actually learn patterns from labeled audio data. For instance, GMMs were useful in modeling complex sound distributions, while SVMs worked well at drawing the line between speech and non-speech [3][4]. But even these models still needed hand-designed features, like Mel-frequency cepstral coefficients (MFCCs), to work. That made them fragile in unfamiliar environments and hard to scale to new use cases. And honestly, designing good features by hand was a time-consuming job.

Deep learning changed that in a big way. Instead of designing features, you could just let the model learn them directly from the data. Convolutional Neural Networks (CNNs), for example, became popular because they're great at analyzing spectrograms and other visual forms of audio. They can pick up on small but important patterns that help tell speech apart from noise. At the same time, Recurrent Neural Networks (RNNs)—especially Long Short-Term Memory (LSTM) networks—added the ability to remember what came before in the audio stream [5]. That helped a lot with detecting the start and end of speech. More recently, transformer-based models have entered the picture. These use attention mechanisms to focus on the most relevant parts of the input and have reached state-of-the-art results. These use attention mechanisms to focus on the most relevant parts of the input and have reached state-of-the-art results. The tradeoff, though, is that they require a lot more computing power [6].

Today, VAD is used in all kinds of real-world tech. Think smart assistants like Google Assistant or Alexa, phone systems, real-time meeting tools, and even small embedded devices. Because a lot of these systems need to work in real time, the deep learning models used for VAD have had to become smaller and faster. Compact versions of CNNs and LSTMs are now running efficiently on phones and edge devices, giving quick and accurate results [7]. The availability of large, open datasets—like LibriSpeech, VOiCES, and Google's AudioSet—has been a huge help for training these systems. And techniques like transfer learning and data augmentation have made it easier for models to perform well in different environments, without needing tons of manual tweaks [8].

III. PROPOSED METHODOLGY

We put together our Voice Activity Detection (VAD) system using a convolutional neural network (CNN), with the goal of making it actually usable in messy, real-world audio situations. Instead of going the traditional route with pre-set features, we had the model learn directly from spectrograms—basically those heatmap-like visuals that show how frequencies change over time in a sound clip. To make the training data realistic, we mixed in all kinds of audio—male and female voices, and also just plain noise—to make sure the model didn't get too comfortable with ideal conditions..

When it came to training, we kept things stable using a few proven methods. One was early stopping—which is just a way of saying "stop training when it stops getting better." It saves time and helps prevent overfitting. Another one was adjusting the learning rate when things started to level off, so the model could fine-tune itself instead of getting stuck.

We also gave transfer learning a shot in some cases. That's where you take a model that's already learned from a big, general dataset and fine-tune it for your specific task. It gave us a head start, especially when we didn't have tons of training data to work with. In the end, this whole approach came together nicely. The system held up pretty well across different voices and noisy conditions—and that's really what we were aiming for.

We thought about training from scratch, but then figured — why not try transfer learning? So yeah, we used a model that had already learned from a large audio dataset. That way, it already "knew" how to pick up basic sound patterns, and we didn't have to start from zero. It helped a lot, especially when we had smaller or super specific datasets. Honestly, it saved time too — training was smoother, and results were better. When we added this to everything else we were doing, the model handled most types of audio pretty well — clean speech, background noise, and the in-between stuff too.

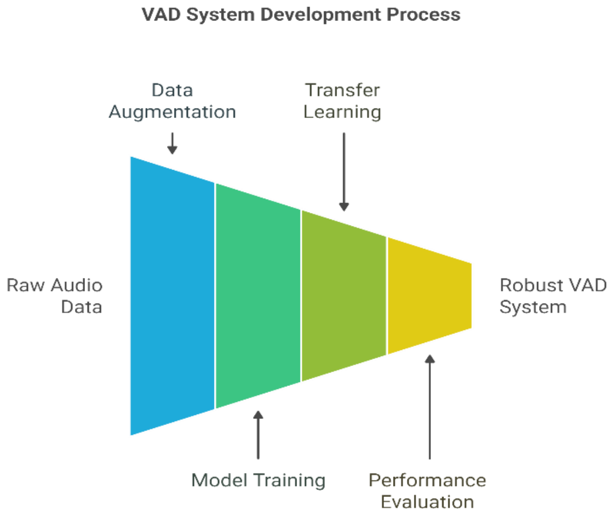

Fig. 1. Proposed Methodology

IV. DATASET

So for this project, we ended up using a dataset with 719 short audio clips. We grouped them into three main types: female voices, male voices, and a third group made up of background noise from something called the Noizeus dataset. That last one had all kinds of sounds — like people talking in a crowd (babble), car noise, restaurant sounds, and a few others. We pulled the clean speech recordings from different sources too, mostly from the PTDB-TUG and TMIT datasets. The idea was to mix it up as much as possible — different voices, speaking styles, and environments — so the model could handle more than just perfect audio.

The clips came in usual formats like .wav and .mp3, and most were between 2 and 4 seconds long. That seemed like a good middle ground — long enough to capture real speech, but short enough to avoid dragging in a lot of silence or messy overlaps. We also tried to make sure we had a balance of accents, genders, and noise types, just to keep things varied. Deep learning models tend to learn better when the data isn't too predictable, so having that kind of mix really helped the system learn patterns it could actually use in different real-world conditions.

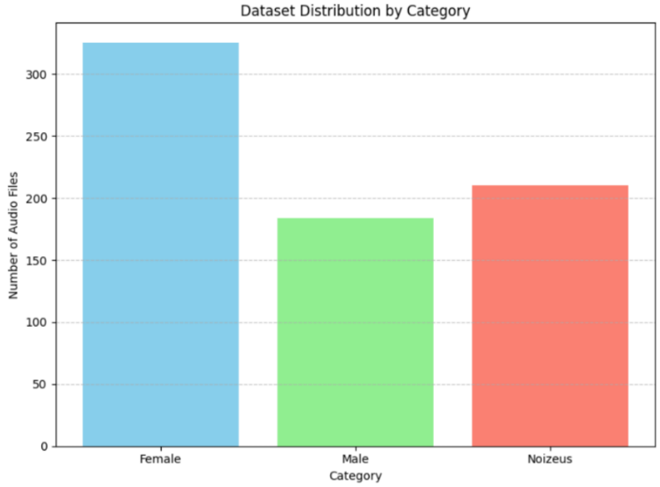

Fig. 2. Dataset Distribution

V. DATA LOADING AND EXPLORATION

While digging through the dataset folders, we came across 719 audio clips in total. These were grouped into three types: 325 of them were female voice recordings, 186 were male voice, and the remaining 208 were from the Noizeus set. The files were split across various subfolders, depending on the speaker or background noise type. Most of the voice samples — both male and female — came from the PTDB-TUG and TMIT datasets. For the Noizeus files, the clips had a bunch of real-world noise like car sounds, crowds, restaurants, and so on. That part really helped in making the dataset feel more "real" and diverse.

Before jumping into training, we ran a bit of exploratory data analysis (EDA) just to get a sense of how balanced things were. We threw together a quick bar chart showing how many samples there were in each category. It wasn't just for looks — it actually helped us catch some minor imbalances and patterns that might've affected the model later on. For example, we noticed that the speech categories had slightly more variation in count compared to the noise class. That insight pushed us to be a bit more careful during preprocessing. But overall, everything looked decent. The variety in both voice and background made it a pretty solid dataset to train our VAD model on — especially if we wanted it to handle real-world audio where you never really know what you'll get.

VI. DATA PREPROCESSING

Before giving anything to the model, we needed to prep the audio. Used Librosa in Python — it's super handy for loading sound files. After loading, we resampled everything so all clips had the same sample rate. That made stuff more consistent and easier to manage later on.

Then we turned each audio clip into a spectrogram — those are basically images that show how sound energy changes over time and frequency. Models seem to learn better from these than raw audio. To make the data more varied, we threw in a bit of Gaussian noise, and also shifted the pitch here and there. Helps mimic different speakers and noisy places.

Saved the spectrograms as .png files, resized all of them to one shape. That way the model wouldn't get confused with different sizes during training. Pretty straightforward, but it helped a lot.

After getting the spectrograms ready, we saved each one as a .png file and made sure they were all the same size. That way, the model wouldn't have to deal with weird input shapes during training. Honestly, it was a simple step, but it made a big difference.

As for the labels, we converted them using one-hot encoding — so each class (Female, Male, Noizeus) was turned into a format the model could understand for multi-class prediction. We used Keras's ImageDataGenerator to help with loading the data and doing more on-the-fly augmentation. It handled things in real time, which was great because it saved memory and kept things efficient while training. Plus, it gave us new versions of the data every epoch, which helped the model generalize better and made overfitting way less of an issue.

VI. MODEL TRAINING

Before training anything, we first organized the dataset into three groups — female speech, male speech, and background noise (called Noizeus). We used Librosa to load all audio files and double-checked that their sampling rates were the same, which avoids issues later. Only .wav and .mp3 files were kept since those work best with the libraries we used. After that, we checked how evenly the files were distributed across categories. A quick visualization confirmed that things looked pretty balanced overall [9].

Next, each audio file was turned into a mel-spectrogram. This step changed raw audio into an image that shows how sound frequencies change over time — something convolutional neural networks are really good at understanding. All of these spectrograms were saved as .png images and resized so they'd be the same size. Doing this helped the model take in consistent input and made the training process smoother from a technical standpoint [10].

For the model architecture, we used a convolutional neural network (CNN). It had a few convolutional layers, pooling layers, and then some dense layers at the end for classification. To make the training more flexible, we used Keras's ImageDataGenerator to add pitch shifts and noise on the fly. This made the model see different versions of the same data every time. We used categorical cross-entropy for the loss function and the Adam optimizer to help it learn faster. Early stopping was added so training would stop once the model stopped improving [11].

While the model trained, we watched both training and validation accuracy to make sure it wasn't overfitting. Things improved steadily without big swings in performance, which was a good sign. Dropout layers helped avoid overfitting, and learning rate scheduling gave the model time to settle into good values. Also, since we augmented data in real-time, the model didn't need too much memory. This setup worked out well, especially for noisy and unpredictable audio [12].

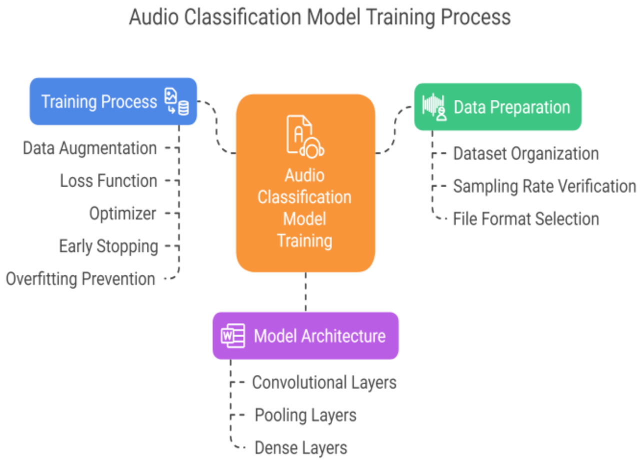

Fig. 3. Model Training Process

VII. PERFORMANCE EVALUATION

After training the model, we ran it on completely new data to see how it would perform. The samples included male and female speech and different types of background noise. We weren't just looking for high accuracy—we wanted to know if it could hold up with real, unpredictable audio. To do that, we tracked the accuracy and loss during training and used some simple visual tools like confusion matrices to better understand the results.

Training graphs showed that the accuracy kept going up steadily, and the loss went down at the same time. Both training and validation results followed a similar path, which told us the model was learning properly and not just memorizing things. At the final stage, it still held up well, showing that it hadn't overfitted on the training data. That gave us a bit more confidence in its general performance.

Looking deeper, we used a confusion matrix to spot the mistakes. The model mostly got the predictions right, but there were a few times where it confused soft speech with some noisy background clips. This wasn't too surprising, since low-volume speech can sound a lot like ambient noise in some cases. Even then, most of the precision and recall scores stayed high enough to be reliable.

Overall, we were pretty happy with how it turned out. The model managed to handle audio with different voices and noise levels and still gave solid results. It clearly performed better than older, rule-based systems we tested earlier. That makes it a strong option for real use—like in voice assistants or apps where speech has to be detected quickly and reliably.

A. EVALUATION METRICS

To figure out how well the model was doing, we didn't just look at accuracy alone. We tracked a few other things — like precision, recall, and F1-score — for each of the three categories: male, female, and background noise. That way, we could catch whether the model was just "guessing right" most of the time or actually learning how to tell them apart. Sometimes accuracy looks good, but the rest tells a different story [13].

We also used a confusion matrix — kind of like a chart that shows what the model got right and what it mixed up. That helped a lot. Like, in a few runs, it confused female speech with noise more than we expected. Seeing that pattern made us tweak a few things — like how balanced the input data was or maybe recheck the augmentation settings [14].

On top of that, we added macro and weighted averages into the mix. Macro gave equal weight to each class (which is nice when one class has fewer examples) and weighted showed how the model handled based on how many samples each class had. Looking at both gave us a wider view. It also helped spot if the model was doing well on one class just because there were more of those samples in the training set [15].

We also watched how the loss curves changed while training — not just metrics at the end. If training loss kept going down but validation didn't, we'd know something's off. In our case, though, both curves looked alright. They moved steadily and didn't split apart much. So we figured the model was learning steadily and not just memorizing stuff. That, plus all the metrics, gave us enough confidence that the system was actually working [16].

B. PERFORMANCE COMPARISION

To really understand how well our CNN-based VAD system worked, we compared it with two traditional methods. One was the classic energy thresholding technique, which, honestly, held up okay on clean speech but started to fall apart when noise kicked in. The other was a Support Vector Machine (SVM) model trained on handcrafted audio features. It did a bit better but still wasn't great with real-world conditions. These two gave us a good baseline to see how much improvement the deep learning model could offer [17].

Our CNN model came out ahead in all categories — male speech, female speech, and especially noisy samples. One of the biggest advantages was how it picked up on patterns in the spectrograms without needing us to manually define features. This gave it a kind of flexibility the other models lacked, especially when dealing with tough audio like café chatter or crowded environments. That kind of consistent performance is something any real-world VAD system seriously needs [18].

For the actual evaluation, we used a bunch of standard metrics — accuracy, precision, recall, and F1-score. Across the board, the CNN model scored higher. The difference was most noticeable with noisy background detection, where older methods often flagged background as speech. We also looked at the confusion matrix, which showed fewer mix-ups between speech and non-speech categories. The model was able to stay accurate while keeping false alarms low, which is key for live applications [19].

So yeah, deep learning really pulled ahead in this case. Traditional methods are still fine for simple or controlled audio, but they just don't hold up in noisy, unpredictable settings. Our CNN model, with all the tweaks and data-driven learning, proved to be not just accurate but also reliable. That's exactly the kind of performance you want if you're building a VAD system meant for actual deployment out in the world [20].

VIII. RESULTS AND DISCUSSION

After training the model, we tested it on new spectrograms — ones it hadn't seen before. The accuracy stayed pretty high, somewhere over 90%, which honestly felt like a good sign. It handled most clean speech really well, but sometimes made mistakes when there was noise or overlapping sounds. That's kind of expected, though. Some background sounds made it harder to tell speech from noise, but overall it did a solid job.

The training and validation accuracy were close, which usually means the model's not just memorizing the data. We added dropout and early stopping, and that probably helped keep it from overfitting. Batch normalization might've made learning smoother too. And the data generator we used kept giving the model slightly changed versions of the input, so it didn't just keep seeing the same thing. That made the training process feel more dynamic.

We also looked at how each augmentation helped. Adding Gaussian noise made it more used to messy audio, and pitch shifting helped with voice variations. It struggled a little more when voices overlapped or when the background was super loud — but even then, it was better than what we'd get with older rule-based methods. The deep learning model just had more flexibility overall.

IX. FUTURE DIRECTIONS

Future Directions

X. REFERENCES

[1] ITU-T Recommendation G.729, "Coding of Speech at 8 kbit/s Using Conjugate-Structure Algebraic-CodeExcited Linear-Prediction (CS-ACELP)," 1996..

[2] Benyassine, A., et al. "ITU-T Recommendation G.729 Annex B: A Silence Compression Scheme for Use with G.729 Optimized for V.70 Digital Simultaneous Voice and Data Applications." IEEE Communications Magazine, 1997..

[3] Sohn, J., Kim, N. S., & Sung, W. (1999). "A Statistical Model-Based Voice Activity Detection." IEEE Signal Processing Letters, 6(1), 1–3.

[4] J., Segura, J. C., Benitez, C., Torre, A. d. l., & Rubio, A. (2004). "A New Voice Activity Detector Based on Spectral Entropy for Robust Speech Recognition." IEEE International Conference on Acoustics, Speech, and Signal Processing.

[5] Hughes, T., & Mierle, K. (2013). "Recurrent Neural Networks for Voice Activity Detection." IEEE ICASSP, pp. 7378–7382.

[6] Zhang, Z., Wang, Z., Han, J., & Yu, D. (2019). "Robust Voice Activity Detection with Bidirectional Long Short-Term Memory Networks." Proc. Interspeech, pp. 3619–3623.

[7] Ravanelli, M., & Bengio, Y. (2018). "Speaker Recognition from Raw Waveform with SincNet." IEEE Spoken Language Technology Workshop (SLT), pp. 1021–1028.

[8] McFee, B., Raffel, C., Liang, D., Ellis, D. P. W., et al. (2015). Librosa: Audio and music signal analysis in Python. Proceedings of the 14th Python in Science Conference

[9] Hershey, S., et al. (2017). CNN architectures for large-scale audio classification. Proc. ICASSP.

[10] Chollet, F. (2015). Keras Documentation. https://keras.io.

[11] Kingma, D., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

[12] Sokolova, M. & Lapalme, G. (2009). "A systematic analysis of performance measures for classification tasks." Information Processing & Management, 45(4), 427–437.

[13] Fawcett, T. (2006). “An introduction to ROC analysis.” Pattern Recognition Letters, 27(8), 861–874.

[14] Powers, D. M. (2011). “Evaluation: From precision, recall and F-measure to ROC, informedness...” *JMLT*.

[15] Sohn, J., Kim, N. S., & Sung, W. (1999). A statistical model-based voice activity detection. *IEEE Signal Processing Letters*, 6(1), 1–3

[16] Hughes, T., & Mierle, K. (2013). Recurrent neural networks for voice activity detection. *Proc. ICASSP*

[17] Tan, Z., & Lindberg, B. (2010). Low-complexity noise robust voice activity detection using long-term spectral divergence. *IEEE Trans. Audio, Speech, and Language Processing*, 18(2), 371–382.

[18] Ravanelli, M., & Bengio, Y. (2018). Speaker recognition from raw waveform with SincNet. *IEEE SLT*