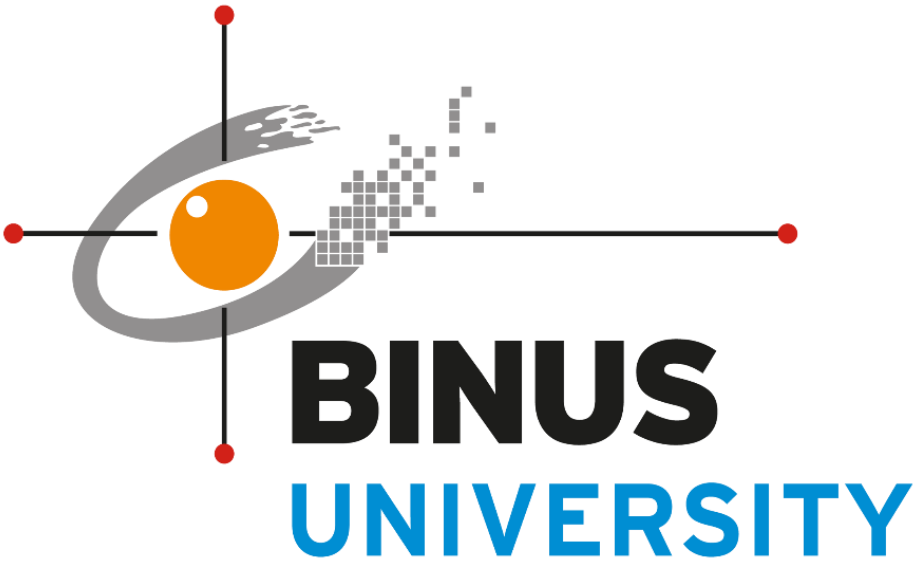
FINAL REPORT DEEP LEARNING (COMP6826001)

RESONAI: REAL-TIME MONOPHONIC VOCAL PITCH ESTIMATION USING DEEP 1D CONVOLUTIONAL NEURAL NETWORKS



Submitted By:

Gilbert Christiansen Tahar – 2702249695

LB01

Lecturer : D5544 – Wawan Cenggoro, S.Kom., M.TI

Computer Science Program

School of Computer Science

Universitas Bina Nusantara

Jakarta

2025

ResonAI : Real-Time Monophonic Vocal Pitch Estimation Using Deep 1D Convolutional Neural Networks

Gilbert Christiansen Tahar
Computer Science Department
School of Computer Science
Bina Nusantara University
Jakarta, Indonesia 11480

Wawan Cenggoro, S.Kom,. M.TI
Computer Science Department
School of Computer Science
Bina Nusantara University
Jakarta, Indonesia 11480

***Abstract*—To detect musical notes(pitch) accurately from a human or indivdual's voice, is the core technology behind the apps like instrument tuners, educational tools that help singers to improve their intonation, or even karaoke scoring. Traditional methods often fails to work when the inputs have background noises, while modern AI models are also often too heavy and slow to be run on a website browser. The main goal of this research is to develop a pitch estimation system or model that balances high accuracy with computational efficiency suitable for real-time use. This research proposes ResonAI, a lightweight Deep learning model that designed to detect the pitch of a single singing voice(monophonic) in real-time. Instead of converting sounds to images like spectrograms, the model uses a 1D-Convolutional Neural Network(1D-CNN) to read raw audio waves directly. The model got trained using the MIR-1K dataset (a collection of 1000 karaoke clips). The results show that the model achieved a best accuracy of 74.39%. The researcher also do an experiment, making the training data artificially harder using data augmentation (adding random background noises and changing volume levels) and actually the result is 69.24% (best accuracy), which is lower than not using data augmentation. This suggests that lightweight model works best on cleaner data. Simply, this research develops a Pitch Estimation model to be implemented as a real-time Vocal Tuner application. The researcher also deployed the model as a real-time web app using Gradio, hosted on Hugging Face Spaces [11]. This implementation demonstrates that the model can be accessed publicly via a standard web browser without requiring any local installation or specific hardware setup. Proving that it's possible to build a fast and functional vocal tuner using a simple AI architecture.**

Keywords—Pitch Estimation, Convolutional Neural Network (CNN), Deep Learning, Real-Time Processing, MIR-1K Dataset, Monophnoic Vocal.

I. INTRODUCTION

Pitch estimation, or the detection of the fundamental frequency (f0) of a sound signal, is a foundational task in Music Information Retrieval (MIR). Accurate pitch tracking is crucial for various apps, such as automatic instrument tuners and melody transcription to educational tools for improving singing intonation and karaoke scoring systems. Traditionally, pitch estimation has relied on mathematical formulas known as Digital Signal Processing (DSP) algorithms. For decades, The YIN algorithm **[1]** and its improved version, **pYIN [2]**, were the standard methods due to their mathematical efficiency. While these older methods are fast, but they have a major weakness, it often fail when there is background noises or the recording quality is poor.

To fix these problems, researchers started using Deep Learning (AI). Dieleman and Schrauwen **[3]** showed that computers can read raw sound waves directly, without needing complex manual processing. Also Khadem-hosseini et al. [4] proved that treating pitch detection as a classification task (choosing the right note from a list) works better than just guessing the frequency number. Following this idea, modern AI models like CREPE **[5]** and Google's SPICE [6] achieved very high accuracy. But these models are usually very large and heavy. The models need powerful computers to run smoothly, which makes them difficult to be used on simple platforms like web browsers or mobile phones (where latency and processing power are the critical constraints).

To bridge or address the gap between high accuracy, efficiency and speed, this research proposes ResonAI, a lightweight AI model designed and trained to detect vocal pitch(monophonic) in real-time. The goal is to develop a model that is smart like modern AI models but fast enough to run instantly on mobile phones or website. The model uses a simple 1D-CNN architecture (4 layers) and was trained on the MIR-1K dataset **[7]**, a collection of singing voice recordings. The model was built and developed using PyTorch [8] for training the AI model and Librosa **[9]** for processing the audio files (audio signal processing).

The main contribution of this project is a streamlined model (end-to-end architecture) that balances performance with speed. To prove that it works and effectiveness in the real world environment and real time, the proposed model is deployed as a cloud hosted vocal tuner web app (fully functional) on Hugging Face Spaces using the Gradio framework [10, 11]. This demonstrates and validates that lightweight AI models can be practical, accessible tools for musicians, singers or everyone without needing expensive or high-end computer hardware and infrastructure.

II. DATASET

A. Dataset and Preprocessing

To develop a robust monophonic pitch estimation model, high quality vocal recordings with accurate ground truth annotations are required. This section outlines the dataset that used by the model. The preprocessing steps is applied to prepare the data for the 1D-CNN architecture, and the augmentation strategies or experiment is also employed.

This research and model used the MIR-1K dataset [7], which stands for "Multimedia Information Retrieval Lab – 1000 clips". This dataset is a popular collection of data for singing voice research. The dataset contains 1000 song clips from 110 chinese pop songs, that sung by 19 amateur singers (11 men and 8 women).

The MIR-1K dataset has a unique feature that is perfect for this project and model. The music and the voices are recorded on separate channels (Left and Right Channel). The left one contains only the singing voice, while the right channel contains only the background music. Since our model and research goal is to detect the pitch of a single voice (monophonic), the model only used the left channel for training the model.

For the data preprocessing part, before giving the audio into the AI model, the researcher had to process the data using some steps. Firstly, do sampling rate, converted all audio files to 16 kHz, it means the audio has 16000 data points per second. Choosing this specific rate because it strikes the perfect balance between speed and quality. While standard music is often recorded at 44.1 kHz, to capture every detail, pitch detection simply does not need that much data. 16 kHz is already high enough to capture the human voice clearly, also low enough to keep the app running fast in real-time. Additionally, the researcher stuck to 16 kHz because it is a standard number in the audio industry. The researcher avoided using random

rates like 17 or 18 kHz because they are non-standard and would likely cause compatibility errors with the software tools, without making the model any smarter.

After that, do slicing or framing, cut the long audio clips into very small pieces that called frames. Each of the frame contains 1024 samples, it's about 0.064 seconds of sound. This size is small enough for the computer to process quickly in real-time. The selected frame size of 1024 samples is to find the best balance between speed and accuracy. If using more larger frames, it would help to detect low notes better but it would make the app feel slow and laggy. On the other side, using smaller frames makes the app faster but it would make the pitch detection got less accurate. Then for the third step, the researcher do label conversion, instead of asking the model to guess the exact frequency number like 404.5 Hz, the researcher converted the pitch into 360 bins or categories. This is a technique that used by the CREPE model [5]. It's like asking the model to choose the correct piano key rather than guessing a precise number, which can help to reduce errors [4].

B. Data Augmentation

To show and see if the model could handle difficult situations, the researcher do experiment using data augmentation to the training data pipeline. It means the data is artificially changed the audio to make it harder for the model to learn. By adding two things, random white noises (static sound) to clean the voice recordings to simulate a noisy environment. The second one is changing the volume, randomly made the audio louder or quieter between 0.8 and 1.2x to stimulate a singer moving closer to or further away from the microphone. The data augmentation was done only when training. When the model do final testing or got deployed, the data augmentation is off. The model receives the original input or voice from the user as it is.

III. MODELING

A. Model Architecture

This section describes the design architecture of the model, the lightweight deep learning model. The traditional models does convert the audio into images (spectrograms) and use heavy 2D CNN's model. The model that the researcher has developed processes the raw audio waveform directly using a 1D Convolutional Neural Network (1D-CNN).

The architecture is designed to be simple and efficient for real time use. It consists of two main parts, Feature Extractor and Classifier. The feature extractor part analyses the raw audio wave to find patterns related to pitch. This uses layers of Convolutional blocks. While the classifier part is to take the patterns found by the extractor and decides which specific musical note or pitch bin is being sung by the user.

For the details of the layer that used by the model. The model takes an input of 1024 samples, representing around 64ms of sound. Then it passes through 4 Convolutional Blocks. Each of the blocks contains the following layers to process the data. Conv1D, Batch Normalization, ReLU (Activation Function), MaxPooling, and Dropout.

The Conv1D is the core layer, scans the audio wave to detect features like peaks and curves that can represent the pitch. Because the layer that the model used is 1D, this requires much less math calculation than the 2D image processing. The Batch Normalization layer adjusts the numbers inside the model to keep them balanced. This layer helps the model learn faster and prevents the model from getting confused during the training loop. The ReLU adds non linearity to the model, allowing the model to learn complex patterns instead of just simple straight lines. This essentially decides which information is important enough to pass to the next layer. The MaxPooling layer is to reduces the size of the data by keeping only the most important features, like the maximum values. This makes the model become lighter and faster without losing any important or critical information. The last one is the Dropout layer, randomly turn off some neurons during the training loop to prevent the model from memorizing the training data, it can cause overfitting and forces the model to learn general rules about pitch. This model used value of 0.4 for the Dropout Layer.

After passing through the convolutional blocks, the data is flattened and passed to a Fully Connected Dense Layer. The final output consists of 360 neurons, that represents the 360 pitch bins, covering 6 octaves. The model uses a softmax function to convert the output into probabilities. Then, the bin with the highest probability is chosen as the predicted pitch.

For the training configuration part, to train the model effectively, the researcher used the following settings, those are Loss Function and Optimizer. The model used Cross-Entropy Loss which is the standard for classification tasks, this calculates the difference between the model's prediction and the actual answer of the correct pitch. Also used the Adam Optimizer, automatically adjusts the learning speed to help the model to find the best solution efficiently. For the epochs and Batch Size, the baseline model was trained for a maximum of 50 epochs and the batch size of 128. But for the model that trained with data augmentation, maximum of 90 epochs and the batch size of 128. Also saved the model weights that achieved the best validation accuracy to prevent overfitting.

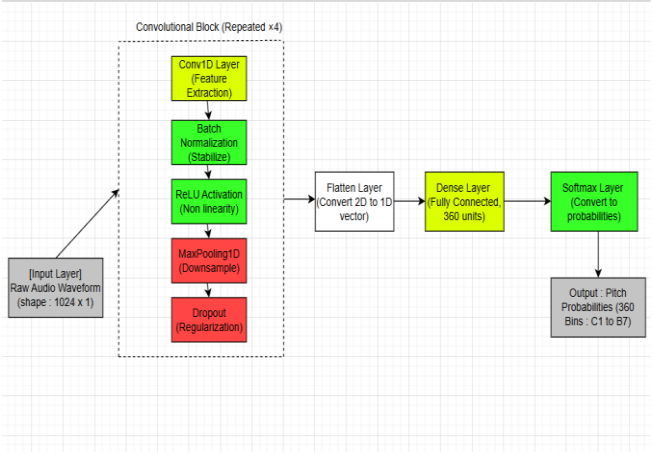


Fig 1. The model design architecture (1D-CNN Architecture).

IV. EVALUATION

A. Performance Overview of the model

This chapter shows, explains and presents the experimental results of the AI model. Also the model's viability for real time apps. The evaluation focuses on assessing the model's pitch estimation and detection accuracy, also analyzing the impact of the data augmentation on its performance.

To measure the performance of the model objectively, for the evaluation metric used Classification Accuracy for the primary metric. Since the problem is formulated as a classification task across 360 output bins (representing pitches from C1 to B7 with 20-cent resolution). The accuracy is calculated as the percentage of audio frames where the model's predicted pitch bin index exactly matches the ground truth target index. Accuracy = Number of Correctly Classified Frames / Total Number of Frames * 100%. A prediction is labeled as correct if and only if the argmax of the model's softmax output vector corresponds precisely to the true categorical label of the input frame.

B. Experimental Results of the model

The model was trained using the MIR-1K dataset and evaluated on unseen data from the test split. The researcher do experiments to compare the performance of the baseline model (trained on clean data) and the model that trained with the data augmentation.

TABLE I. ACCURACY OF THE BASELINE MODEL (CLEAN DATA) AND THE MODEL WITH DATA AUGMENTATION

	Baseline Model (50 Epochs)	**Model With Data Augmentation (90 Epochs)**
Accuracy	74.39%	69.24%

From Table I, the baseline model achieved the highest accuracy of 74.39%. The result shows that the lightweight 1D-CNN architecture is capable of learning meaningful pitch representations from raw audio waveforms in monophonic singing context. Compared to the model with the data augmentation, the accuracy is lower than the baseline model. The model is designed to be lightweight with relatively few parameters, has limited model capacity. The model struggles to handle complex data that created by augmentations (such as adding noises) and failed to learn properly. The minimal network could not effectively separate the clear singing voice from the added background noise. This shows a significant trade-off, while lightweight models are fast, they are less able to handle complex data variations compared to bigger, heavier models like CREPE [5].

```
Model Saved!
Epoch [29/50] | Loss: 0.8369 | Train Acc: 73.72% | Val Acc: 73.70%
Model Saved!
Epoch [30/50] | Loss: 0.7821 | Train Acc: 75.39% | Val Acc: 73.61%
Epoch [31/50] | Loss: 0.7268 | Train Acc: 77.15% | Val Acc: 73.61%
Epoch [32/50] | Loss: 0.6824 | Train Acc: 78.53% | Val Acc: 74.13%
Model Saved!
Epoch [33/50] | Loss: 0.6360 | Train Acc: 80.01% | Val Acc: 73.43%
Epoch [34/50] | Loss: 0.5946 | Train Acc: 81.31% | Val Acc: 73.78%
Epoch [35/50] | Loss: 0.5571 | Train Acc: 82.57% | Val Acc: 73.67%
Epoch [36/50] | Loss: 0.5225 | Train Acc: 83.71% | Val Acc: 73.21%
Epoch [37/50] | Loss: 0.4929 | Train Acc: 84.53% | Val Acc: 74.39%
Model Saved!
Epoch [38/50] | Loss: 0.4625 | Train Acc: 85.55% | Val Acc: 73.82%
Epoch [39/50] | Loss: 0.4401 | Train Acc: 86.21% | Val Acc: 73.99%
Epoch [40/50] | Loss: 0.4136 | Train Acc: 87.09% | Val Acc: 72.85%
Epoch [41/50] | Loss: 0.3966 | Train Acc: 87.63% | Val Acc: 73.21%
Epoch [42/50] | Loss: 0.3777 | Train Acc: 88.28% | Val Acc: 73.48%
Epoch [43/50] | Loss: 0.3591 | Train Acc: 88.83% | Val Acc: 73.71%
Epoch [44/50] | Loss: 0.3405 | Train Acc: 89.36% | Val Acc: 73.82%
Epoch [45/50] | Loss: 0.3272 | Train Acc: 89.81% | Val Acc: 73.66%
Epoch [46/50] | Loss: 0.3149 | Train Acc: 90.23% | Val Acc: 73.39%
Epoch [47/50] | Loss: 0.2994 | Train Acc: 90.69% | Val Acc: 73.51%
Epoch [48/50] | Loss: 0.2888 | Train Acc: 91.03% | Val Acc: 73.57%
Epoch [49/50] | Loss: 0.2778 | Train Acc: 91.38% | Val Acc: 73.70%
Epoch [50/50] | Loss: 0.2683 | Train Acc: 91.64% | Val Acc: 73.40%
Best Validation Accuracy: 74.39%
```

Fig 2. Training loop of the baseline model over 50 epochs

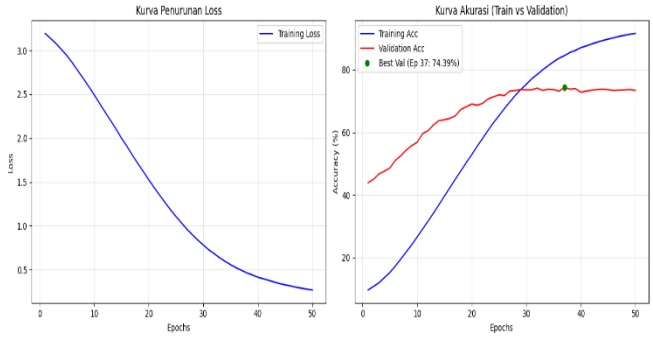


Fig 3. Training Curves for the Model (clean data) : (a) Training Loss, (b) Training and validation accuracy over 50 epochs.

Figure 3 shows the training curves loop of lightweight 1D-CNN architecture. The left curve shows a consistent decrease in training loss, indicating successful learning convergence. The right curve compares training (blue) and validation (red) accuracy. The model achieved its best validation accuracy of 74.39% at epoch 37 (marked with a green dot). The divergence between the training and validation curves beyond epoch 37 indicates the onset of overfitting, therefore, the model weights from this epoch were selected for the final model.

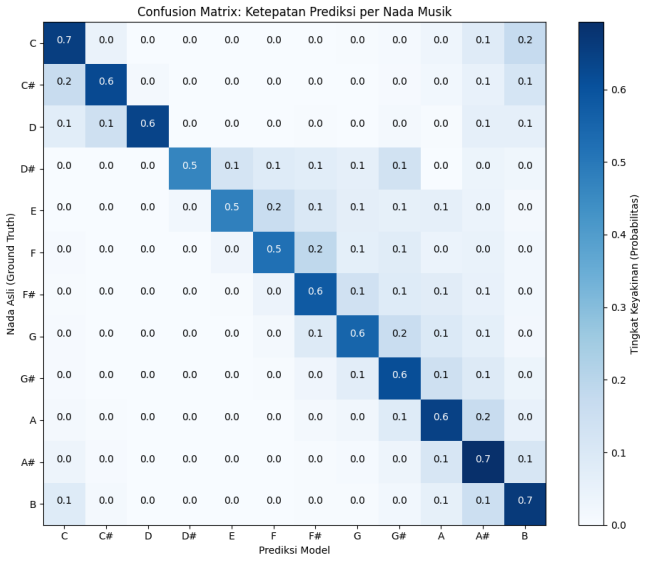


Fig 4. Confusion Matrix of the Model.

The confusion matrix in Fig. 4 presents the normalized confusion matrix evaluating the classification performance of the model. across the 12 musical semitone classes (C to B). The vertical axis represents the ground truth pitch, while the horizontal axis indicates the model's prediction. The values within each cell and the accompanying color bar on the right indicates the prediction probability (proportion), with darker blue telling higher confidence levels. The analysis of the confusion matrix shows that the model performs well. This is evident from the highest values consistently appearing along the main diagonal line, means that the model most frequently predicts the correct pitch for each class. The accuracy rates for correct pitches range from 0.5 to 0.7.

The model's error pattern is also clear and consistent. The majority of classification errors occur only in adjacent semitones (one note higher or lower). For example, for the true pitch 'D', the model correctly predicts it 60% of the time, and the errors that

occurs are predictions of 'C#' (0.1) or 'D#' (0.1). The model almost never incorrectly predicts pitches that are far apart.

But there is slight variation in performance between pitches (for instance, 'C' and 'A#' have an accuracy of 0.7, while 'D#', 'E', and 'F' have an accuracy of 0.5), overall, this matrix confirms and tells that the model understands pitch structure well, where the errors that occur are generally only minor semitone deviations.

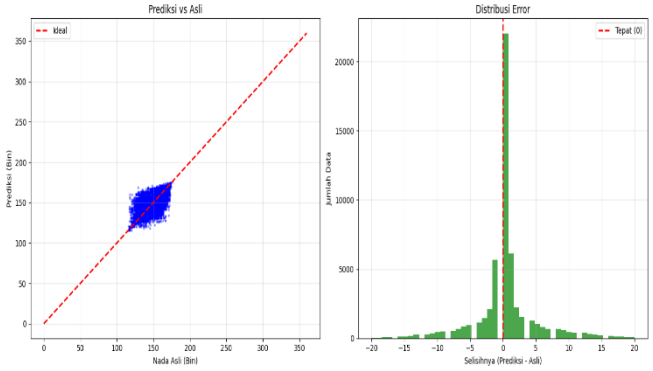


Fig 5. (a) Prediction vs real answers labels plot (scatter plot), (b) Distribution Error of the model (baseline model).

Figure 5 evaluates the prediction accuracy on the test set. The scatter plot on the left shows that most predictions (blue dots) cluster tightly around the ideal red line, indicates the predicted pitch is usually very close to the actual pitch. The error distribution on the right confirms this, showing a sharp peak at zero. This means the majority of predictions are exactly correct, and most errors are very small deviations.

C. Discussion on Error and Improvement Opportunities

The experimental results demonstrate that the model architecture can effectively learn pitch representations directly from raw audio waveforms, achieving a peak validation accuracy of 74.39%. The analysis of prediction errors shows that misclassifications are mostly confined to nearby semitones, indicates a solid understanding of musical structure.

But a significant finding is the model's limited capacity due to its lightweight design. The training curves indicated a tendency to overfit after epoch 37. Furthermore, attempts to improve robustness using standard data augmentation techniques negatively impacted performance. This suggests a trade-off: while the model is computationally efficient for real-time use, its small size makes it less resilient to complex variations and noise compared to larger, heavier models.

To address these limitations and enhance the model in future research, several strategies can be explored. Architectural Adjustments, to handle complex data and augmentation better, the model capacity could be slightly increased. This might involve adding more convolutional filters or layers, balanced with stronger regularization techniques (such as increased dropout rates) to prevent overfitting.

Dataset Expansion, the current model was trained on the MIR-1K dataset. Training on larger, more diverse datasets that include various instruments, vocal styles, and real-world background noises could improve general robustness. Also Post-Processing, implementing a post-processing step, such as smoothing predictions over time using a median filter or Hidden Markov Model (HMM), could help correct isolated errors and make the output pitch contour more stable.

V. DEPLOYMENT

This section details the practical implementation of the trained ResonAI model into a functional, real-time prototype app. The objective of this deployment phase is to move the model from a purely experimental environment into a publicly usable and accessible interface, demonstrating its capability to process live audio input from a user's microphone and provide instantaneous pitch feedback. To achieve it, the system is deployed as a cloud hosted web app on Hugging Face Spaces [11]. This approach will make sure that the app is universally accessible via any standard web browser on laptops or mobile devices. It eliminates the need for users to possess high-end hardware or install specialized software like Python or GPUs.

The deployment architecture consists of the trained model backend (the 1D-CNN model that performs the actual mathematical calculations to guess the pitch), the interactive frontend (a user interface that handles microphone access, streams audio data to the backend, and visualizes the pitch feedback or predicted musical note in real-time).

Before deployment, the trained model must be saved in a portable format. The trained model also acts as the sytem's brain. Based on the evaluation results in Chapter IV, the researcher selected the model weights from epoch 37, which achieved the best balance between accuracy and generalization (avoiding overfitting).

The researcher utilized PyTorch's serialization mechanism (torch.save) to export the model's state dictionary into a standalone binary .pth file (vocal_tuna_best.pth). This file essentially acts as the frozen brain of the app, containing all the learned parameters (weights and biases) necessary for inference, ready to be loaded instantly when the application starts. Not like the training phase which relied on mounting external storage from Google Drive for deployment, the model file is bundled directly within the app's root directory. This standalone structure allows the app to be self sufficient and portable, capable of running on any server environment without broken file path dependencies.

To rapidly develop a robust and interactive prototype, the researcher utilized Gradio [10], an open-source Python library specifically designed for creating machine learning demos. Gradio was selected because of ease of use (Gradio simplifies the complex process of handling real-time audio streaming from a web browser microphone) and real time capability (supports live interfaces where the output updates instantly as the input changes). The app is also hosted on Hugging Face Spaces [11] using a basic CPU environment. While the model relies on cloud processing, the lightweight nature of the model will make sure that the latency will remains low enough for a near real-time user experience.

The main core of the deployment is the real-time processing loop that manages the data flow from the input user microphone to prediction display. Firstly, audio capture (raw audio input from the microphone and automatically resampling it to the required 16 kHz sampling rate), frame buffering (continuous audio stream got sliced into discrete frames of 1024 samples each, 64ms

of audio), direct interface (takes the raw 1024 sample frames directly as input), prediction and display (outputs the probability scores for 360 pitch bins. An argmax operation identifies the bin with the most highest probability. The bin index is instantly mapped to its corresponding musical note names like C, D, E. And displayed the output on the screen for the user). This whole entire flow or pipeline repeats many times per second, creating a seamless and real time experience for a user singing into the microphone.

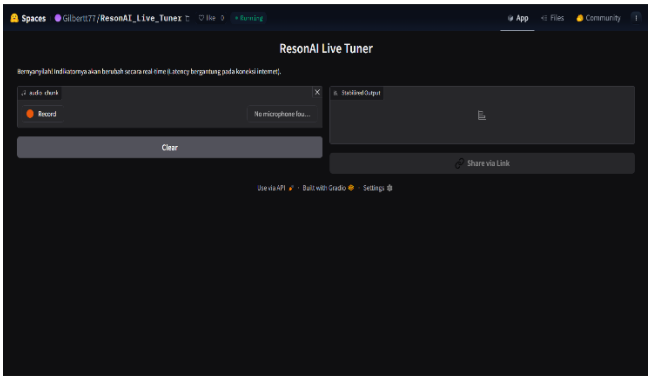


Fig 6. Screenshot of the ResonAI web interface running live on Hugging Face Spaces.

CONCLUSION

This research successfully developed a lightweight deep learning model based on a 1D-Convolutional Neural Network (CNN) for real-time pitch estimation effectively. Compared to traditional methods that rely on complex preprocessing, This model processes raw audio waveforms directly, making it computationally efficient and suitable for real-time web and mobile apps.

The experimental results on the MIR-1K dataset showed that the model achieved a peak validation accuracy of 74.39%. Analysis of the results confirmed that the model successfully learned musical structures, with most errors confined to nearby semitones (the neighbor notes) rather than random guesses.

A major milestone of the project was the successful deployment of the model to a cloud-hosted web interface (Hugging Face Spaces). This practical implementation validates the researcher primary hypothesis, that a lightweight AI model can effectively and fully functional in a real-time environment, accessible via standard web browsers, without requiring high-end hardware.

However, a key finding is the existence of a significant trade-off. While the model's lightweight design ensures high speed, it results in limited model capacity. This was evident as the model began to overfit after epoch 37 and struggled to handle complex data introduced by standard augmentation techniques. In conclusion, the model architecture offers a viable solution for speed-critical apps but requires careful tuning to manage its limited capacity against complex data.

To overcome current limitations and improve the performance, future research can focus on some areas like Diverse Datasets (training the model on larger, more varied datasets beyond MIR-1K Dataset to improve its generalization to different instruments and real-world environmental noises), Balancing Capacity (investigating ways to slightly increase the model size or complexity to handle noisy data better without sacrificing too much real-time speed), and Post Processing (implementing temporal smoothing techniques, such as median filtering, on the final output to stabilize the predicted pitch contours and correct isolated errors).

REFERENCES

[1] A. de Cheveigné and H. Kawahara, "YIN, a fundamental frequency estimator for speech and music," *The Journal of the Acoustical Society of America*, vol. 111, no. 4, pp. 1917–1930, Apr. 2002. doi : https://doi.org/10.1121/1.1458024

[2] M. Mauch and S. Dixon, "pYIN: A fundamental frequency estimator using probabilistic threshold distributions," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 659–663. doi : 10.1109/ICASSP.2014.6853678

[3] S. Dieleman and B. Schrauwen, "End-to-end learning for music audio," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2014, pp. 6964–6968. doi : 10.1109/ICASSP.2014.6854950

[4] M. Khadem-hosseini, S. Ghaemmaghami, A. Abtahi, S. Gazor, and F. Marvasti, "Error correction in pitch detection using a deep learning based classification," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 990–999, 2020. doi : 10.1109/TASLP.2020.2977472

[5] J. W. Kim, J. Salamon, P. Li, and J. P. Bello, "CREPE: A Convolutional Representation for Pitch Estimation," in *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2018, pp. 161–165. doi : 10.48550/arXiv.1802.06182

[6] B. Gfeller, C. Frank, D. Roblek, M. Sharifi, M. Tagliasacchi, and M. Velimirović, "SPICE: Self-supervised Pitch Estimation," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 1118–1128, 2020. doi : 10.1109/TASLP.2020.2982285

[7] C.-L. Hsu and J.-S. R. Jang, "On the improvement of singing voice separation for monaural recordings using the MIR-1K dataset," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 2, pp. 310–319, Feb. 2010. doi : 10.1109/TASL.2009.2026503

[8] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, 2019, pp. 8024–8035. doi : 10.48550/arXiv.1912.01703

[9] B. McFee *et al.*, "librosa: Audio and Music Signal Analysis in Python," in *Proceedings of the 14th Python in Science Conference*, 2015, pp. 18–24. doi : 10.25080/Majora-7b98e3ed-003

[10] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, and J. Zou, "Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild," *arXiv preprint arXiv:1906.02569*, 2019. doi : 10.48550/arXiv.1906.02569

[11] Hugging Face, "Hugging Face Spaces Documentation," 2025. [Online]. Available: https://huggingface.co/docs/hub/spaces . [Accessed: Dec. 14, 2025].