COMPLETE
Journal of Computer, Electronic, and Telecommunication
ISSN 2723-4371, E-ISSN 2723-5912

Article

Fan Speed Level Control Using Three-Language Voice Commands Based on YAMNet Audio Classification in *Deep Learning*

Giga Razki Arianda[1], Heri Pratikno[2*], Musayyanah[3], Pauladie Susanto[4]

[1,2,,4] Computer Engineering, Faculty of Technology and Informatics, Universitas Dinamika, Surabaya, Indonesia
[3] Politeknik Elektronika Negeri Surabaya;
* Correspondence: heri@dinamika.ac.id

Received: 28 October 2025; Revised: 26 December 2025; Accepted: 30 December 2025; Published: 31 December 2025

Abstract: The process of interaction between humans, computers, and electronic equipment can now be made more interactive, natural, and intuitive. In several previous studies, this interactive process was carried out through sensors or detection of finger gestures using computer vision based on MediaPipe. In this research, we designed and built a system that can control the fan rotation speed level using voice commands from three languages, namely Indonesian, English, and Javanese in real time through an audio classification process with YAMNet. The research results in the training process with 15 epochs had 100% accuracy, loss 0.46, ROC curve class 0 (fan off) was 100%, class 1 (low rotation fan) was 100%, class 2 (medium rotation fan) was 99%, and class 3 (high rotation fan) was 100%. Meanwhile, the results of testing the subset test dataset model using 15 epochs for all commands produced a percentage value of 97.5%.

Keywords: artificial intelligence; audio classification; arduino uno; deep learning; voice recognition

1. Introduction

The process of interaction between humans and computers, machines, and electronic devices was initially invasive or required physical contact. However, with the advancement of technology, this interaction process no longer requires physical contact or is non-invasive thanks to the support of various sensors, including motion sensors [1], sensors [2], sound sensors [3], and so on. The support of these various sensors enables the interaction process between humans and existing electronic equipment to be carried out automatically and more responsively, but it is not yet natural.

In study [4], a simulation of an automatic fan control device was created using a fuzzy system and a web-based DHT11 temperature sensor. With the massive development of new technologies to assist and facilitate people's daily work in various fields of life, this can now be done by applying soft computing technology in computer vision. The application of human-computer interaction technology in computer vision using the MediaPipe framework [5] through the process of detecting hand gestures to turn a fan on and off locally has been achieved. Computer vision research on lip reading for Indonesian word recognition: a viseme-based approach to assist people with hearing disabilities [6].

The application of a method that can be used to reduce the feature dimensions of data is Principal Component Analysis (PCA), and the Computed Input Weight Extreme Learning Machine (CIW-ELM) classification method [7] is used to classify simple activities [8] such as walking, climbing stairs, descending stairs, sitting, standing, and lying down. Another study [9] related to a system for

detecting the shapes of both the right and left hand finger gestures to automatically control two fans in a local area also uses MediaPipe.

In further research [10], it is possible to control the operation of a fan remotely through computer vision by detecting the shape of the right hand gesture using MediaPipe with the application of Internet of Things (IoT) technology based on the MQTT (Message Queue Telemetry Transport) protocol. Next, the application of MediaPipe and CNN (Convolutional Neural Network) for finger gesture detection can be used as a counting medium for early childhood education [11]. Conventional control of electronic devices through voice commands using the Arduino Uno R3 microcontroller has also been carried out [12]. Real-time classification of gunshots, glass breaking, and speech using YAMNet on edge computing networks [13].

Prior research on fan speed control employed hand gesture recognition using MediaPipe with the fan's three factory-installed speed settings. This study advances the field by developing an AI-driven system that controls fan speed through multilingual voice commands (Indonesian, English, and Javanese) using real-time audio classification via YAMNet in a deep learning environment. A key distinction from previous work is the novel speed control mechanism: rather than relying on preset buttons, this system achieves variable speed control through calculated AC voltage division using capacitor capacitance, with validation via AVOM measurements.

2. Materials and Methods

The deep learning intelligent system computation in this research was programmed on the Arduino Uno IDE interface platform, Visual Studio Code program, and Jupyter using the Python programming language. Figure 1 shows that the input data comes from a laptop microphone to recognize the user's voice. The input is then processed by a computer using audio classification with the YAMNet model. The output is sent to the Arduino Uno microcontroller to perform actions on the relay by turning off or adjusting the fan speed based on the voice detection process in the three trained languages.

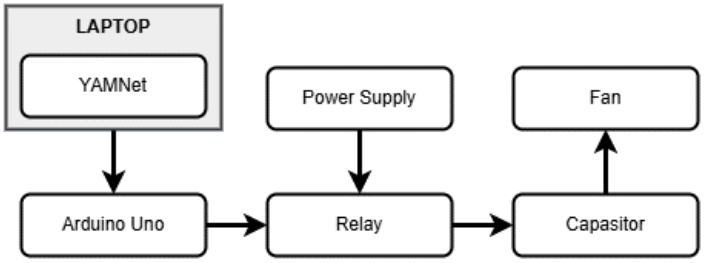


Figure 1. System block diagram

The schematic diagram of the system is shown in Figure 2. The hardware design process involves selecting specifications and adjusting various elements, such as sensors, microcontrollers, and other supporting components. Each part of the hardware is carefully arranged and connected to ensure synergistic and efficient work. The main objective of this hardware design model is to support the implementation of experiments and data collection accurately and consistently in this study.

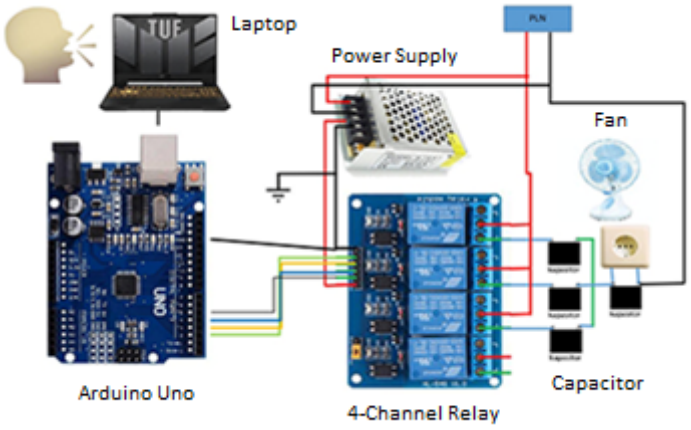


Figure 2. Schematic diagram of the system

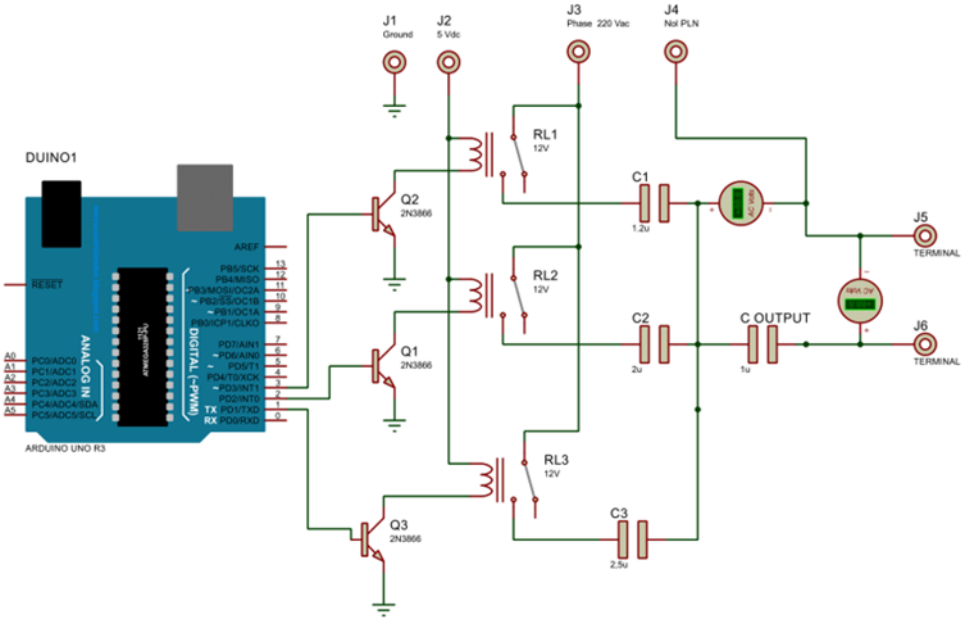


Figure 3. Schematic diagram

From the circuit schematic in Figure 3, there are several relay channels connected to the Arduino Uno microcontroller port. Each relay is connected to a capacitor with a different capacitance value to regulate the fan speed level. Channel one has a capacitor with a capacitance of 1.2 uF, channel two has a capacitor of 2 uF, and channel three has a capacitor of 2.5 uF. These capacitors will later be connected in series with a 1 uF output capacitor. The function of the output capacitor is to act as a voltage divider for the voltage supplied to the fan. A voltmeter is used to monitor the voltage across the capacitors connected in series. To determine the capacitive reactance of each capacitor, use equation (1). The formula for capacitive reactance is Xc.

$$Xc = \frac{1}{2 \times \pi \times f \times C} \tag{1}$$

2.1. Installation Environment

The environment installation stage is divided into two parts consisting of several *libraries* used in the sound classification process, such as *TensorFlow Hub*, *soundfile*, *numpy*, *pandas*, and other *libraries* needed during the dataset training process. *TensorFlow Hub* is one of *the libraries* from *TensorFlow* that

provides various models that can be accessed and used more easily. In this study, *the* main *library* contains the YAMNet model, which can be called up via *the link "https://tfhub.*dev/google/yamnet/1". *TensorFlow Hub* facilitates access to *pre-trained* models that have been uploaded to the *platform*. By utilizing YAMNet via *TensorFlow Hub.* This research can quickly and efficiently integrate advanced audio classification capabilities into the system being developed. *The Serial library* is used to connect to the Arduino Uno microcontroller from Python.

In this study, the early stopping method was applied to the training process of the YAMNet model for fan command classification based on audio signals. Early stopping is a regularization technique used in the training process of deep learning models to prevent overfitting. This method works by automatically stopping the training process when the model's performance on the validation data no longer shows significant improvement after a certain number of epochs. Thus, the model is not forced to continue learning until it reaches a predetermined maximum number of epochs.

2.2 Dataset Flowchart

In this study, the author created *an audio dataset* based on commands used to control a fan using three languages, namely: Indonesian, English, and Javanese. Figure 4 below shows *the flowchart* of the stages of *the* audio *dataset* creation process. The first step is to collect audio data recordings in the form of command words desired in this study. Then, labels are given to each audio data and initial processing of the audio data is carried out before use, such as changing the audio format, such as WAV or MP3, normalizing *the volume*, or removing background *noise*.

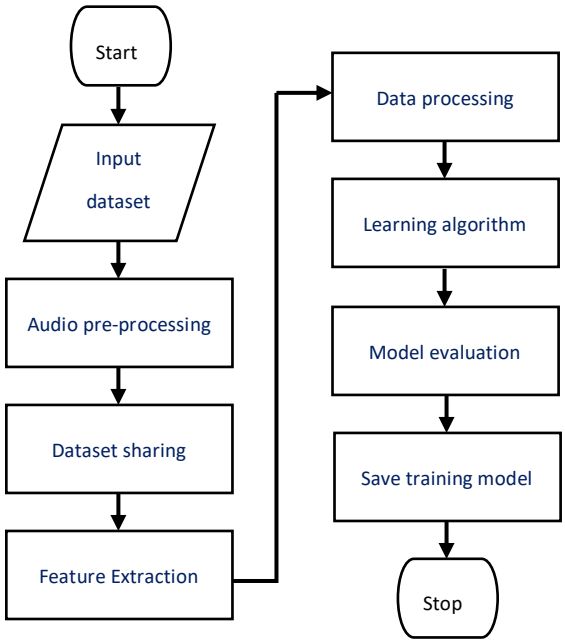


Figure 4. Flowchart of audio dataset

The audio dataset will be divided into several parts, including a training set, a validation set, and a test set. Next, using the processed training data, during the training process, the model will learn to recognize patterns in the sound data. The validation stage is used to set the data model parameters, such as the number of layers in the neural network. The test set is used to evaluate the model's ability to calculate matrices such as accuracy, precision, recall, and F1-score, which are parameters used to measure how well the model performs in voice recognition. After modeling, if the data model performs well, it can be used for voice recognition on new data.

2.3 Dataset Collection

The audio dataset used in this study consists of human voices uttering commands in three different languages. To turn off the fan in all three languages, the same root keyword "fan" is used, while to adjust the fan speed in all three languages, the same root keyword "fan button" is used, as shown in Table 1. The collected dataset consists of 270 audio files from 18 people, with four commands from each language and five repetitions of each command.

Table 1. Multilingual voice commands for fan speed control system

Language	Fan Off	Speed 1 (Slow)	Speed 2 (Medium)	Speed 3 (High)
Indonesian	kipas mati	kipas tombol satu	kipas tombol dua	kipas tombol tiga
English	kipas off	kipas tombol one	kipas tombol two	kipas tombol three
Javanese	kipas pejah	kipas tombol siji	kipas tombol loro	kipas tombol telu

2.4 Dataset Augmentation

In this study, the researchers developed a strategy to increase the amount of data to improve the performance of the system, namely: using data augmentation techniques to enlarge the data set by applying high pass filters, medium pass filters, and low pass filters on the collected sound samples. Augmentation with high pass, medium pass, and low pass channels provides frequency variations in the sound dataset, unlike low pass filters which ensure constant frequencies, high pass filters are used to ensure consistent frequencies in the air. Medium pass channels create a middle ground between the two. This aims to make the model more adaptive to acoustic environment variations that may occur in everyday life.

By applying sound information augmentation through high pass, medium pass, and low pass channels, the author seeks to improve the robustness and generalization of the model to variations that may occur in different environmental conditions. This composition is expected to help the model better recognize and understand various sound variants in the sound classification process. The final result of the data augmentation process is a total collection of 720 files from 4 commands in 3 predetermined languages, as shown in Figure 5.

Figure 5. Dataset augmentation process

2.5 Audio Characteristics

At this stage, it is crucial to consider the characteristics of the audio being used, which requires visual monitoring using *waveforms* and *spectrograms* involving visual representations of audio signals. This is essential for a deep understanding of the unique features contained in the audio data. Through *waveform* visualization, monitoring of amplitude fluctuations over time can be done clearly, providing an overview of basic dynamics, including duration, tempo, and amplitude variations that may affect overall audio characteristics, as shown in Figure 6.

Meanwhile, spectrograms provide further information by presenting the distribution of frequency energy in the audio signal over time. Using spectrum frequency this, monitoring changes in frequency, *pitch*, and harmonic elements can be achieved. The combination of visual *waveform* and spectrogram creates a comprehensive approach to analyzing and understanding audio characteristics. This stage plays a key role in data preparation, enabling this research to recognize underlying patterns in audio data before entering the *encoding* and model development stages.

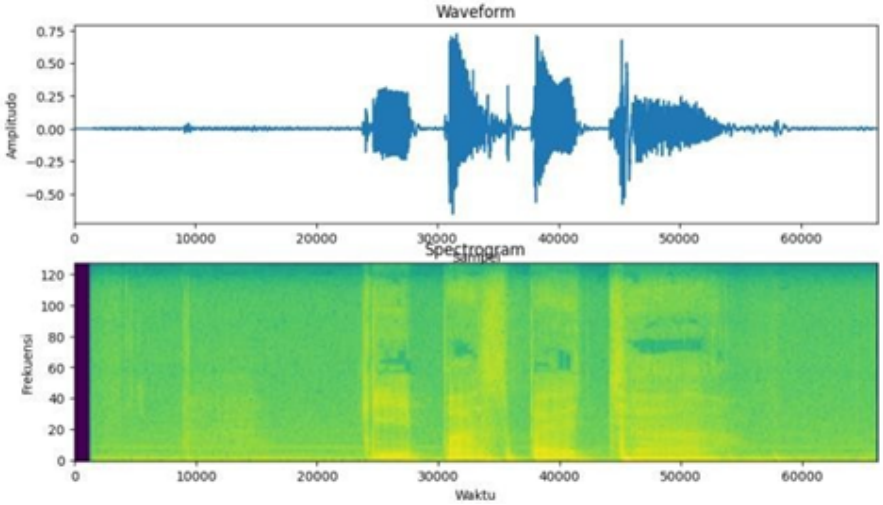


Figure 6. Waveform and spectrogram

2.6 Data Clustering

It is necessary to group the data into four categories of fan speed, namely off, low, medium, and high. In each category *folder*, the audio data is then grouped based on language. The researcher created *subfolders* for each related language, namely Indonesian, English, and Javanese, with the aim of creating a more detailed and organized structure, facilitating the management of the dataset for each fan speed *level* using the language used in the spoken commands, as shown in Figure 7.

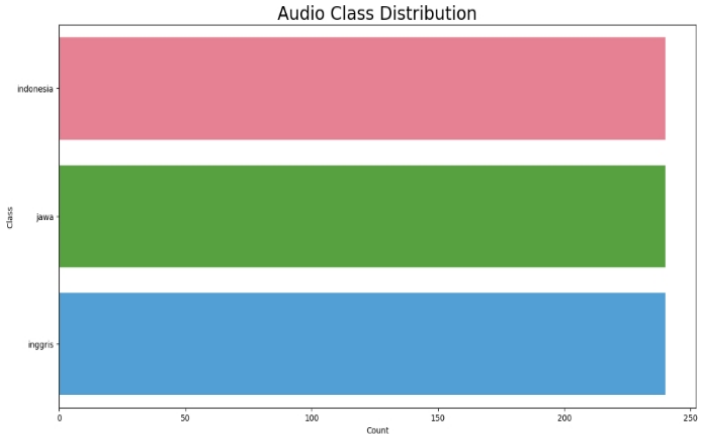


Figure 7. Data grouping based on language

2.7 Data Category Distribution

The next step is to distribute the dataset into five *folders* named *folds*, namely *folds* 1, *folds* 2, *folds* 3, *folds* 4, and *folds* 5. This distribution is carried out according to *the datasheet* that has been *randomly* generated by the program. The main purpose of distributing the data into *folds* is to ensure that each *fold* contains a balanced representation of various languages and categories of fan speed commands, as shown in Figure 8.

In each *fold*, the dataset will consist of *the categories* "off," "low," "medium," and "high." Each of these categories represents a different level of sound intensity. This grouping is done to ensure that each *fold* covers *the range of* fan speeds that may be encountered in practical use. By distributing the dataset proportionally across *folds*, researchers hope to achieve better model generalization and be able to recognize a wider variety of sounds. This process is a very important part of dataset preparation before entering the model training and evaluation stage.

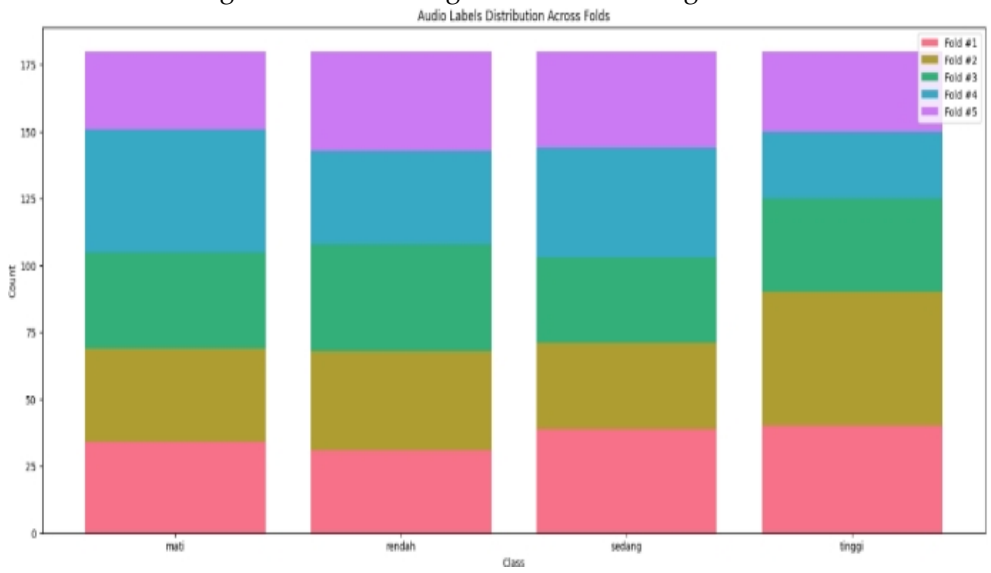


Figure 8. Data distribution based on category

2.8 Folds Subset Distribution

Folds are a *cross-validation* technique used to evaluate the performance of machine learning models. In *folds*, the data is divided into several parts called *folds*, as shown in Figure 9. The model is then trained on several *folds* and tested on the remaining ones. This process is repeated several times with each *fold* being used as *a* test *fold* once. The *folds* technique has several advantages over other *cross-validation* techniques, namely:

- More accurate in estimating model performance on unknown data
- More stable because it does not depend on the order of data used for training

In the YAMNet model, *folds* are used to overcome *overfitting*. *Overfitting* is a problem that occurs when the model is too well-suited to the training data that it cannot produce good results on unknown data. By using folds, the YAMNet model can be trained on larger data sets, thereby reducing the risk *of overfitting*.

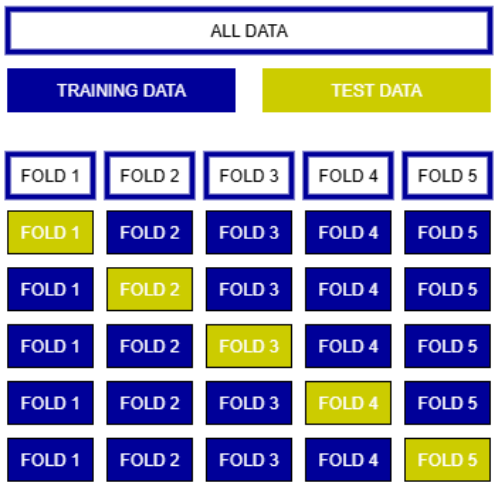


Figure 9. Dataset *folds*

The fold distribution stage divides the dataset into three main *subsets*: *train*, *test*, and *validation*, as shown in Figure 10. *Folds* 1 and *folds* 3 are selected to be part of *the train* data, while *folds* 4 and *folds* 5 are allocated as *validation* data. Finally, *folds* 2 are designated as *test* data. This division aims to optimize model training using *train* data, objectively test model performance using previously unused *test* data, and validate model results on *validation* data to prevent *overfitting*. Each *subset* has a different number of datasets. *The train subset* has 287 audio datasets, *the validation subset* has 279 audio datasets, and *the test subset* has 154 audio datasets, so that from a total of 3 *subsets* there are 720 audio datasets, as shown in Figure 11.

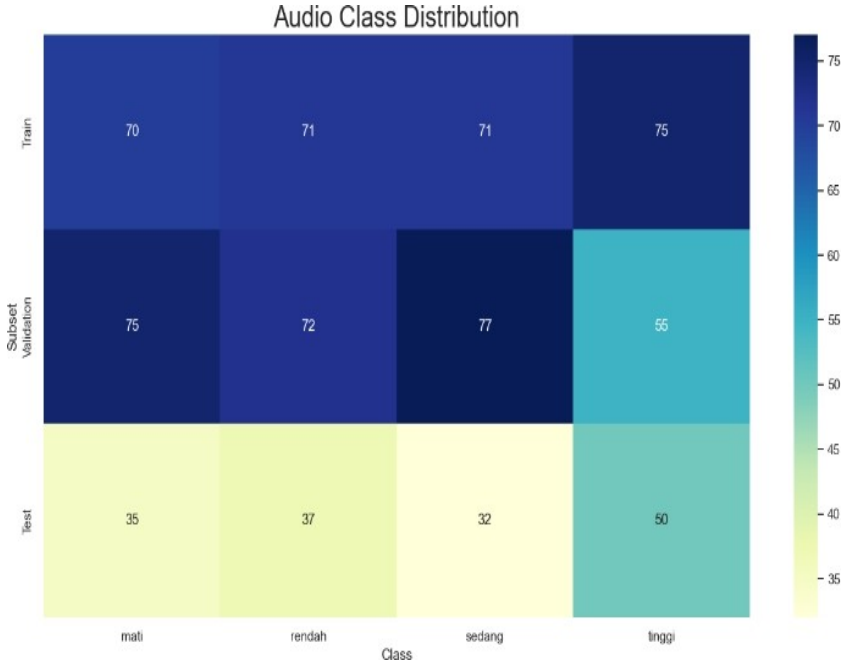


Figure 10. Distribution of *subset folds*

Dataset Split Sizes

	Subset	Number of Samples
0	Train	287
1	Validation	279
2	Test	154
3	Total	720

Figure 11. Subset dataset table

2.9 Label Encoding

The next step is to perform *encoding* on the audio labels, which aims to convert the audio *class* labels into numerical representations in three *subsets* consisting of *train, test*, and *validation*. This *encoding* is a crucial stage in audio data processing, with the main objective being to convert audio *class* information into numerical values that can be understood by machine learning models. This process is performed on each data *subset*, including *train* to train the model, *test* to test the model's performance, and *validation* to validate the learning results, the label *encoding* process is shown in Figure 12.

```
# Get label encoded audio classes
train_labels_encoded = train_df["classID"].to_numpy()
val_labels_encoded = val_df["classID"].to_numpy()
test_labels_encoded = test_df["classID"].to_numpy()

# Inspect label encoded targets
train_labels_encoded
```

[23] ✓ 0.0s

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3], dtype=int64)
```

Figure 12. Label *encoding*

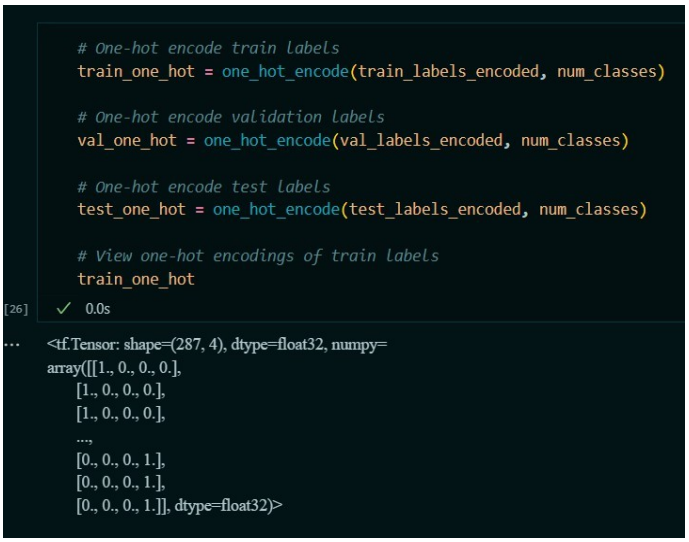


Figure 13. Label one-hot encoding

2.10 Label One-Hot Encoding

Still in the *class encoding* stage, but at this stage it is done using the *one-hot encoding* method as shown in Figure 13 above. The purpose *of one-hot encoding* is to translate data labels into binary vectors. This method focuses on simplifying class representation in the context of *classification* tasks. When involving audio data in *the train, test*, and *validation subsets,* each *class* is represented by a binary vector with a length corresponding to the number of *classes*. In this context, vector elements have a value of one indicating the *class* represented, while other elements have a value of zero.

Comparison of the *one-hot encoding* method, *Gower distance* combined with the *k-means*, DBSCAN, and OPTICS algorithms, and *k-prototype* for mixed-type data clustering. The dataset used in this study [14] is a chronic kidney disease (CKD) dataset sourced from UCI. *Machine Learning Repository*. Based on evaluation using the silhouette index, it is known that *k-prototype* with a *cluster* count of k=2 is the most optimal *clustering* method because it provides the highest silhouette index value compared to the other four methods. When linked to the labels in the dataset, the *clustering* results provide an accuracy value of 81.25 percent.

2.11 BiLSTM

Deep Learning using LSTM in this study [15] was used to detect the Indonesian Sign Language System (SIBI) as a means of communication with the deaf. Research [16] applied LSTM to recognize finger gestures as a means of input for interaction between humans and computers in assessing the level of satisfaction with public customer service using *computer vision*. *Bidirectional Long Short- Term Memory* has two LSTM networks, the first of which functions to process input data sequences in *a forward direction,* and the second LSTM network functions to process data sequences in the opposite direction (*backward). The output* from the *forward* and *backward* LSTM networks is combined at each time sequence. With these two opposing *layers*, the model can learn past and future information for each *input sequence* [17]. The BiLSTM architecture is shown in Figure 14 [18].

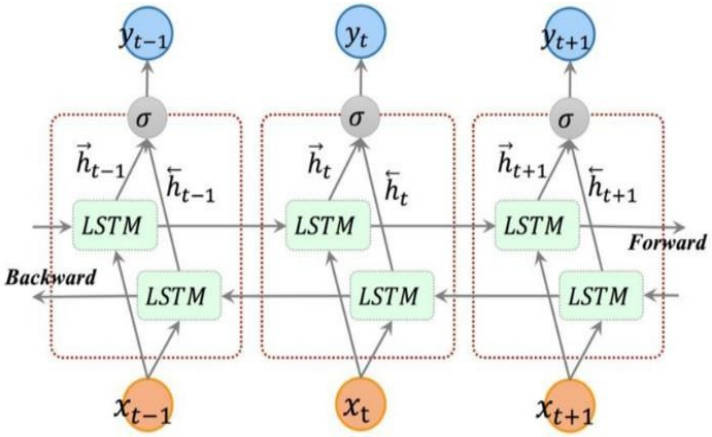


Figure 14. BiLSTM Architecture

2.12 YAMNet Model

After preparing the dataset for training the model, the main focus is on the *layer* structure to be used. This process involves the use of *an* audio *generator*, a tool used to generate *embedding* and label pairs from audio files by providing paths and labels that have been *encoded* using *one-hot encoding*. Next, an input pipeline is built using the *tf.data* API by utilizing paths and labels for audio data. This pipeline includes steps such as defining dataset output tags, creating a *frame* dataset with *a generator*, applying randomization based on certain conditions, enforcing batching, and applying *caching* and *prefetching* based on conditions.

The next step is to divide it into relevant *subsets* for training, testing, and validation. This division is crucial for accurately testing the model's performance. After the dataset is divided, the next stage involves analyzing the sample and label forms to ensure compatibility with the YAMNet audio classification model. Model initialization begins by defining *the* kernel *initializer* to be used. Then, *the* input *layer* is formed with *a shape of* 4,1024 and the data type used is *tf.float32* in *the* LSTM *layer*. A *bidirectional* approach with 128 *initializers* is used to improve the model's ability to capture the temporal context of the audio data.

At this stage, *a dropout* of 0.25 is applied to *the* audio LSTM *layer*. *Dropout* is used to reduce *overfitting* by randomly ignoring some units during training. This helps improve the model's generalization to previously unseen data. The use of *dropout* in *the* audio LSTM *layer* aims to optimize the overall performance of the model with the model structure shown in Figure 15. The final step is to calculate the classification probability in the model. This process allows the model to provide predictions for the categories that have been identified during the training process. In this way, it is possible to evaluate the extent to which the model is able to understand and recognize unique audio patterns. The YAMNet model with the designed program involves the use of *a classifier* model in H5 format, which is the result of dataset training.

Provides voice *input* through the program a pre-designed voice recording program. When the program detects voice *input* from the user, it saves the data to *a file* newly in *.wav* format with *a sample rate* of 16,000 Hz. Once the user is detected, the program will predict important categories based on the trained model. For example, if the user matches the patterns "fan off," "fan off," and "fan broken," the model will predict the label "off." Similarly, for the patterns "fan button one," "fan button one," and "fan button siji," the model will predict the label "low." Meanwhile, for the patterns "fan button two," "fan button two," and "fan button loro," the model will predict the label "medium." If the patterns are "fan button *three*," "fan button three," and "fan button telu," the model will predict the label "high."

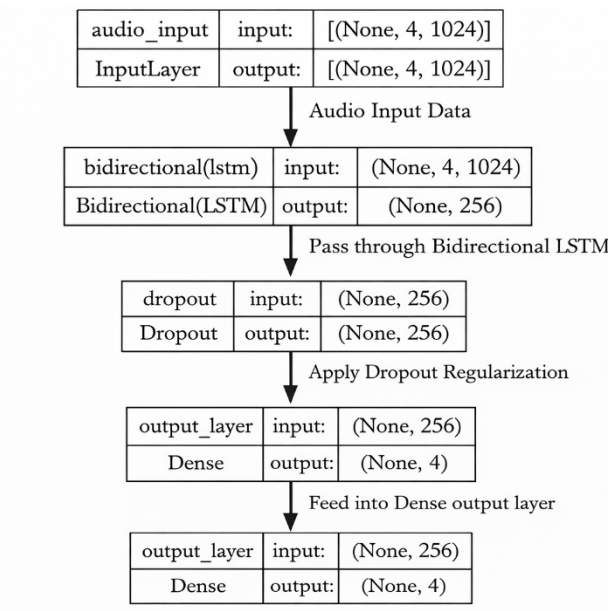


Figure 15. *Dropout* LSTM model structure The user

2.13 Fan Control Algorithm

An overview of the system workflow can be seen in Figure 16, starting with the use of a microphone as input to detect sounds in three languages with twelve different commands. The detected sounds include instructions such as "fan off," "fan button one," "fan button two," "fan button three," "fan pejah," "fan button one", "fan button two", "fan button three", "fan *off"*, "fan button *one"*, "fan button *two"*, and "fan button *three"*. Next, the detected sound will be processed through a grouping step to categorize it into the labels "Off," "Low," "Medium," and "High." The results of the sound recognition will be displayed as detected command sentences along with their category labels.

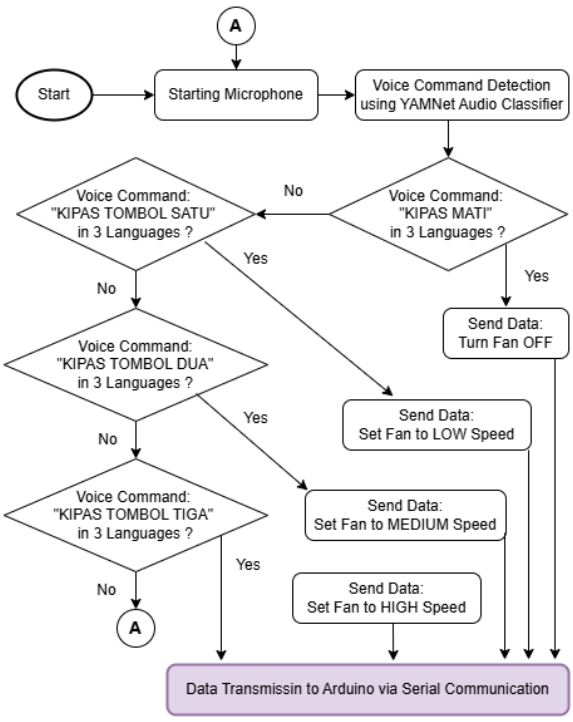


Figure 16. Python *Flowchart*

After processing the data in the Python program using audio classification, the recognition results will be sent to Arduino Uno via serial communication, as shown in Figure 17. On Arduino Uno, the received data will be checked to ensure that the labels generated match the user's commands. If the label received is "*off,"* all *relays* will be *turned off*, so the fan will be turned off. If the label received is "low," *relay 1* will be activated, so the fan will operate at level one or low speed.

For the label "medium", *relay* 2 will be activated, so that the fan is at level two or medium. Finally, if the label received is "High", then *relay* 3 will be activated, so that the fan operates at a speed of level three or high. Next, the system will return to the voice recording application to start the next voice recognition cycle. With the proper integration between the Python workflow and Arduino Uno, the system can provide an effective response according to the voice commands given by the user.

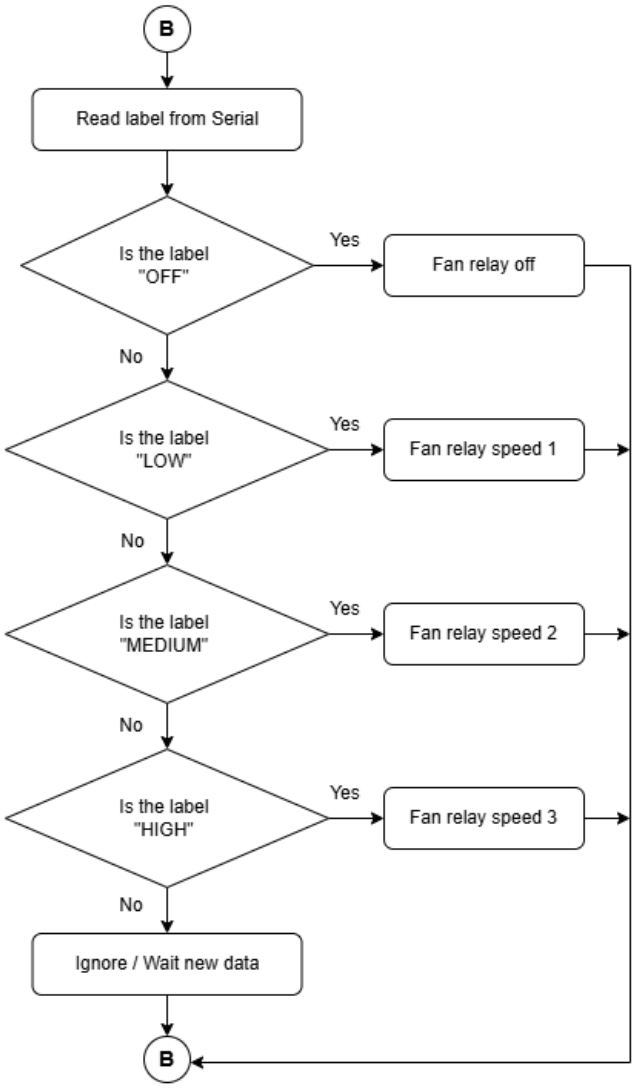


Figure 17. Arduino Uno *Flowchart*

3. Results and Discussion

In this study, to observe the performance of the training process, test parameters such as accuracy percentage and loss value were used, which were obtained from the analysis of the confusion matrix, ROC Curve, classification report, and performance metrics (*accuracy, precision, recall, F1-Score*). All of the above parameters were applied to the three different model training processses, namely: model training with 12 epochs, 15 epochs, and early stopping model training.

3.1. Comparison of Test Results Between Epochs

Based on the test results conducted using 12 *epochs*, 15 *epochs*, and the *early stopping* method (training accuracy will automatically stop at a certain *epoch* because the system sees that the accuracy is already at its highest, close to 100%, so the next *epoch* computation process is no longer necessary). When applying the *early stopping* method, the YAMNet model tends to classify the sound of a fan that is turned off as the sound of a fan that is turned on. This occurs because the *early stopping* model has already achieved a high accuracy percentage at *the* fifth *epoch*, resulting in insufficient knowledge. In general, the training process with 15 *epochs* provides better accuracy for all fan commands, except for the "off" and "stop" commands, *the* 12th *epoch* provides better accuracy.

Based on Table 2, the true accuracy for all fan commands increases with the number of *epochs*, but the true accuracy for the "off" and "stop" commands in the *early-stopping epoch* actually decreases compared to training using 15 *epochs*. This shows that the YAMNet model in the *early-stopping* model more often classifies the sound of a "dead" fan as the sound of a "live" fan. A possible cause is that the YAMNet model is overtrained on the training data, making it too sensitive to the patterns in the training data. As a result, the YAMNet model finds it more difficult to classify fan sounds that are different from the training data.

Table 2. Comparison of user accuracy contained in the dataset between epochs

Command Word	12 *Epoch*		15 *Epoch*		*Early Stopping*	
	Accuracy (%)					
	True	False	True	False	True	False
Kipas mati Kipas *off* Kipas pejah	10	90	40	60	53,3	46,7
Kipastombol satu Kipas tombol *one* Kipas tombol siji	30	70	43,3	46,7	43,3	56,7
Kipas tombol dua Kipas tombol *two* Kipas tombol loro	56,7	43,3	50	50	36,7	63,3
Kipas tombol tiga Kipas tombol *three* Kipas tombol telu	20	80	40	60	43,7	56,7

Table 3 presents a comparison of system accuracy in recognizing voice commands from users not included in the training dataset (unseen users) across three different scenarios: 12-epoch training, 15-epoch training, and early stopping. The evaluation was conducted on 12 voice commands in three languages (Indonesian, English, and Javanese), measuring True Positive and False Positive rates for each command. Results indicate that system performance varies across scenarios. In the 12-epoch training, the command "Kipas tombol two" achieved the highest accuracy (60% True, 40% False), while "Kipas off" had the lowest accuracy (43.3% True, 56.7% False). The 15-epoch training showed improvements in some commands, such as "Kipas off" increasing to 60% True, though some other commands experienced decreased performance. The early stopping scenario provided more balanced results, with accuracy ranging from 50% to 66.7% for the True category.

Overall, no single scenario consistently outperformed across all commands, indicating that the model exhibits varying sensitivity to the number of training epochs for different command classes. These findings suggest the need for further optimization in hyperparameter selection and training strategies to enhance model generalization to new users. The "off" and "stop" fan voice commands do not always have the same pattern because they can be influenced by various factors, including: the type of fan, the size of the fan, and environmental conditions. The YAMNet model, which is

overtrained on the fan patterns found in the dataset, will find it more difficult to classify the "off" and "stop" fan commands that do not have the same patterns as those found in the dataset.

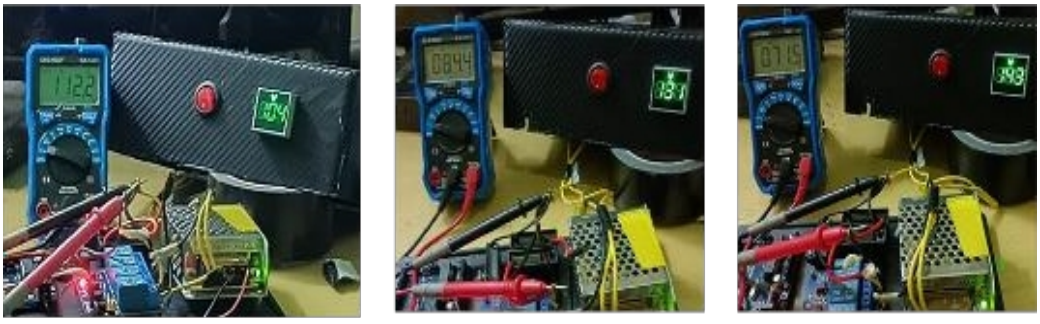
Table 3. Comparison of accuracy of users not in the dataset between epochs

Command Word	12 *Epoch*		15 *Epoch*		*Early Stopping*	
	Accuracy (%)					
	True	False	True	False	True	False
Kipas mati Kipas *off* Kipas pejah	43,3	56,7	60	40	56,7	43,3
Kipas tombol satu Kipas tombol *one* Kipas tombol siji	40	60	60	40	66,7	33,3
Kipas tombol dua Kipas tombol *two* Kipas tombol loro	60	40	60,7	33,3	56,7	43,3
Kipas tombol tiga Kipas tombol *three* Kipas tombol telu	43,3	56,7	56,7	43,3	50	50

3.2. Capasitor Voltage Measurement

To validate the theoretical circuit design, voltage values calculated using Equations (1) and (2) were compared with voltmeter measurements (Figure 18). This validation quantifies the accuracy of the capacitor-based voltage division method at three speed levels. Results are presented in Table 4 (individual capacitors) and Table 5 (series configuration). This verification confirms that the calculated capacitance values achieve the target voltage levels required for precise fan speed control, demonstrating the viability of continuous speed regulation versus discrete preset buttons.

$$\%error = \frac{|Avo - Teori|}{Teori} \times 100\% \quad (2)$$



(a) (b) (c)

Figure 18. Voltage measurement (a) low *level*, (b) medium *level* and (c) high level

Table 4. Capacitor voltage measurement after relay

No.	Capasitor	Experimental Result		Percentage Difference in Voltage (%)
		Voltmeter (VAC)	Theory (VAC)	
1	Xc_1 (1,2 *uF*)	112,2	100	12,2
2	Xc_2 (2 *uF*)	84,4	73	15
3	Xc_3 (2,5 *uF*)	71,5	61	17

Table 5. Series capacitor voltage measurement

No.	In Series		Experimental Result		Percentage Difference in Voltage (%)
	Input Capasitor	Output Capasitor	Voltmeter (VAC)	Theory (VAC)	
	Xc1 (1,2 *uF*)	1 *uF*	104	120	13
	Xc2 (2 *uF*)		131	146	10,3
	Xc3 (2,5 *uF*)		143	159	10,1

The percentage results of capacitor voltage measurements in a series circuit between the capacitor after *the relay* (*input* capacitor) and the 1 *uF* capacitor (*output* capacitor) show some differences between the measured values and the theoretically calculated values. For capacitor Xc1, the voltmeter measurement shows a voltage of 104 VAC, while the theoretical result should be around 120 VAC. The difference between these two values results in a percentage of 13%. Similarly, for capacitor Xc2, the measured voltage is 131 VAC, while the value obtained based on theoretical calculations is 146 VAC, with a percentage of 10.3%.

Similarly, for capacitor Xc3, the measured voltage was 143 VAC, while the value obtained based on theoretical calculations was 159 VAC, with a percentage of 10.1%. The difference between the measurement and calculation results indicates the possibility of certain factors that can affect the measured voltage value, such as imperfections in the device circuit, internal component resistance, or the actual condition of the capacitor. Thus, the experimental results provide important information for a better understanding of the system's response in real situations.

5. Conclusions

Based on the results and discussion of the empirical research conducted in the previous chapter, the following conclusions were drawn: the *training* process with 15 *epochs* had an accuracy of 100%, *a loss* of 0.46, ROC *curve class 0* (fan off) of 100%, *class 1* (low fan speed) of 100%, *class 2* (medium fan speed) of 99%, and *class 3* (high fan speed) of 100%, which is the highest compared to the *12-epoch training* process and the *early stopping* model. High fan speed produced *a classification report* with a *precision* value of 100%, a *recall* value of 100%, and an *F1-score* value of 100%.

The test process of the test dataset subset using the early stopping model, which was conducted 40 times on all commands, had an accuracy of 100%. Meanwhile, the test process using the early stopping model with live commands through a laptop microphone from 3 languages with the fan off (0) had an accuracy of 56.7%, low fan speed (1) had an accuracy of 66.7%, medium fan speed (2) had an accuracy of 56.7%, and high fan speed (3) had an accuracy of 50%. The results of testing with voice commands directly into the microphone did not reach 70% accuracy.

References

1. A. Hanafie, Kamal, and R. Ramadhan, "Perancangan Alat Pendeteksi Gerak Sebagai Sistem Keamanan Menggunakan ESP32 CAM Berbasis IoT," *J. Teknol. dan Komput.*, vol. 2, pp. 142–148, 2022, doi: 10.56923/jtek.v2i02.101.
2. A. A. Syukron and Isnaini Lilis Elviyanti, "Pembuatan Sensor Cahaya dengan Memanfaatkan LED dan LDR Berbasis Arduino Uno," *J. Kridatama Sains Dan Teknol.*, vol. 3, no. 02, pp. 161–169, 2021, doi: 10.53863/kst.v3i02.435.

3. Budy and T. Radillah, "Sistem Kontrol Menghidupkan Lampu Otomatis Menggunakan Sensor Suara FC-04 Berbasis Arduino Uno," *Indones. J. Comput. Sci.*, vol. 12, 2023, doi: 10.33022/ijcs.v12i1.3121.
4. N. K. Daulay, N. Lestari, and A. Armanto, "SIMULASI MONITORING PENGATUR KECEPATAN KIPAS ANGIN MENGGUNAKAN SISTEM FUZZY BERBASIS WEB," 2020, [Online]. Available: https://api.semanticscholar.org/CorpusID:225411605.
5. M. A. Fakhruddin, "TA : Sistem Deteksi Gestur Jari Tangan menggunakan Mediapipe dan Faster-RCNN untuk Mengontrol Kecepatan Kipas Angin," Universitas Dinamika, 2023.
6. H. Wicaksono, L. Liliana, and A. N. Tjondrowiguno, "Pemodelan Lip Reading Bahasa Indonesia Berbasis Visem Menggunakan VGG16 serta Jaro-Winkler Similarity dan Bigram," *J. Infra*, 2022, [Online]. Available: https://publication.petra.ac.id/index.php/teknik-informatika/article/view/12513%0Ahttps://publication.petra.ac.id/index.php/teknik-informatika/article/download/12513/10814.
7. M. Irwanto, F. Bachtiar, and N. Yudistira, "Klasifikasi Aktivitas Manusia Menggunakan Algoritme Computed Input Weight Extreme Learning Machine dengan Reduksi Dimensi Principal Component Analysis," *J. Teknol. Inf. dan Ilmu Komput.*, vol. 9, p. 1195, 2022, doi: 10.25126/jtiik.2022965504.
8. F. D. Tanugraha, "TA : Sistem Pengenalan Aktivitas Manusia Menggunakan Long Short-Term Memory dan Mediapipe," Universitas Dinamika, 2022.
9. Y. R. B. Edowai, "TA : Sistem Automatic Feature Selection Berbasis Deteksi Gestur Kedua Jari Tangan untuk Mengontrol Level Kecepatan Putaran 2 Kipas Angin menggunakan Mediapipe," Universitas Dinamika, 2023.
10. F. Wakerkwa, "TA : Kontrol Level Kecepatan Putaran Kipas Angin melalui Deteksi Bentuk Gestur Jari Tangan Berbasis IoT," Universitas Dinamika, 2023.
11. M. R. P. Nautica, "TA : Hand Gesture Detection sebagai Alat Bantu Ajar Berhitung menggunakan Mediapipe dan Convolutional Neural Network secara Realtime," Universitas Dinamika, 2022.
12. A. A. Firmansyah, "Rancang Bangun Alat Bantu Penyandang Disabilitas Tangan Untuk Menghidupkan dan Mematikan Perangkat Elektronik Menggunakan Voice Recognition Module V3," *J. Telecommun. Netw. (Jurnal Jar. Telekomun.*, vol. 3, no. 2 SE-, pp. 47–52, Nov. 2016, doi: 10.33795/jartel.v3i2.220.
13. C. Malmberg, "Real-time Audio Classification on an Edge Device-Using YAMNet and TensorFlow Lite," 2021.
14. Z. Fadilah and A. W. Wijayanto, "Perbandingan Metode Klasterisasi Data Bertipe Campuran: One-Hot-Encoding, Gower Distance, dan K-Prototype Berdasarkan Akurasi (Studi Kasus: Chronic Kidney Disease Dataset)," *J. Appl. Informatics Comput.*, vol. 7, pp. 57–67, 2023, doi: 10.30871/jaic.v7i1.5857.
15. F. X. L. Riberu, "TA: Sistem Deteksi Simbol pada SIBI (Sistem Isyarat Bahasa Indonesia) secara Real Time menggunakan Mediapipe dan LSTM," Universitas Dinamika, 2023.
16. B. G. Permana, "TA : Sign Language Detection sebagai Alat Bantu Survey Pelayanaan Publik Menggunakan Long Short Term Memory Secara Realtime," Universitas Dinamika, 2023.
17. D. I. Puteri, "Implementasi Long Short Term Memory (LSTM) dan Bidirectional Long Short Term Memory (BiLSTM) Dalam Prediksi Harga Saham Syariah," *Euler J. Ilm. Mat. Sains dan Teknol.*, vol. 11, no. 1, pp. 35–43, 2023, doi: 10.34312/euler.v11i1.19791.
18. Z. Cui, R. Ke, Z. Pu, and Y. Wang, "Deep Bidirectional and Unidirectional LSTM Recurrent Neural Network for Network-wide Traffic Speed Prediction," pp. 1–11, 2018, [Online]. Available: http://arxiv.org/abs/1801.02143.