Proceedings of EKONBIZ 2025
ISBN: 978-99955-45-50-5
UDK: 007.52:681.53]: 658.5
Submitted: 10.04.2025.
Accepted: 20.0.2025.
Review a Book, Instrument, Computer Program, Case, Scientific Event

MODEL FOR VOICE RECOGNITION IN MANUFACTURING PROCESSES

Saša Sudar
College of Applied Studies „Sirmium“, Sremska Mitrovica, Serbia
sasa.sudar@gmail.com
ORCID: 0009-0002-2601-2993

Zdravko Ivanković
College of Applied Studies „Sirmium“, Sremska Mitrovica, Serbia
ivankovic.zdravko@gmail.com
ORCID: 0009-0003-4044-6445

Srđan Damjanović
Faculty of Business Economics Bijeljina, University of East Sarajevo, Republic of Srpska, Bosnia and Herzegovina
srdjan.damjanovic@fpe.ues.rs.ba
ORCID: 0000-0003-4807-5311

Luis Silva Rodrigues
CEOS.PP, ISCAP, Polytechnic of Porto, Portugal
lsr@iscap.ipp.pt
ORCID: 0000-0001-5471-8083

Abstract: *The subject of this paper is the presentation of a practically implemented speech recognition model capable of distinguishing words based on artificial intelligence. The paper provides a detailed explanation of the application of a given voice recognition algorithm, implemented using standard deep and convolutional neural networks, the Python programming language, and machine learning libraries Keras and TensorFlow. This machine learning model recognizes several words using neural networks. The core concept of the presented model is the transformation of sound into images (log-spectrograms), leveraging lessons learned from image recognition to identify spoken words (audio recordings). The described voice control model was developed for the needs of the meat processing industry. Primarily, this model was designed for voice-based data entry from livestock ear tags at the slaughterhouse reception. The goal of implementing this model is to reduce human error in manual data entry and facilitate the overall adoption of the information system.*

The benefits of such a system are numerous. First and foremost, it would increase the speed and efficiency of every worker in all processes, thereby improving overall production. Additional benefits include enhanced tracking of information flow and a reduced risk of errors. Moreover, this system would indirectly contribute to workplace safety by reducing the number of injuries that often occur due to the removal of protective equipment, which complicates computer operation or the completion of paper documents. The proposed system allows computers to be removed from production facilities where climate and working conditions are challenging.

Key words: *artificial intelligence, neural network, model, speech recognition, manufacturing*

JEL classification*: O33*

1. INTRODUCTION

The subject of this paper is the presentation of a practically implemented speech recognition model capable of distinguishing spoken words based on artificial intelligence. The paper provides a detailed explanation of the application of the given voice recognition algorithm, implemented using standard deep and convolutional neural networks,

the Python programming language, and the machine learning libraries Keras and TensorFlow.

Google, YouTube, Sony, and other renowned global companies have developed their voice recognition models using artificial intelligence. Their models are quite complex and extensive, as they are designed to recognize a vast vocabulary of a language. These systems require powerful computers capable of processing large amounts of data in a short time interval.

Our machine learning model is designed for recognizing a smaller set of words using neural networks. The core concept of the presented model is the transformation of sound into images (log-spectrograms), leveraging lessons learned from image recognition to identify spoken words (audio recordings).

The described voice control model was developed for the needs of the meat processing industry. Primarily, this model was designed for voice-based data entry from livestock ear tags during the reception of livestock at the slaughterhouse. The goal of implementing this model is to reduce human error in manual data entry and to facilitate the acceptance of the information system by production workers as a whole. Most slaughterhouse workers have limited IT knowledge and often resist using computers for data entry and processing.

The described system could also be used for voice-commanded autonomous guided vehicles (AGVs) in meat production, replacing human labor in transporting meat to cold storage rooms. Once a worker fills the crates with meat, they can issue a voice command to the AGV robot to transport the pallet to a specific position in a designated cold storage chamber. Cold storage facilities for meat preservation, due to their extremely low temperatures, have adverse effects on workers' health. Employees who frequently enter and exit these storage rooms experience drastic temperature variations of over 50°C, leading to frequent sick leave and even permanent reductions in work capacity, which poses a significant challenge for employers in managing human resources.

The benefits of such a system are numerous. Firstly, it would increase the speed and efficiency of workers in all processes, thus improving overall production. Additional benefits include better tracking of information flow and a reduced likelihood of errors during data entry. The proposed system would also indirectly enhance workplace safety by reducing work-related injuries. In slaughterhouses, worker injuries often occur due to the removal of protective equipment, which makes computer operation or paperwork completion difficult.

The presented system enables the elimination of personal computers from various production facilities where climatic and working conditions are challenging. Data entry commands in the production process can be given by voice, while data storage and management of distributed ERP systems are handled by a voice recognition system utilizing artificial intelligence.

Although this model was developed for the meat industry, we believe it can be successfully applied in many other industries. We consider that this model can be implemented in all workplaces where workers are exposed to high or low temperatures, dangerous chemicals, various contaminants, and where recording necessary data on paper or inputting data into a computer is particularly difficult or dangerous to the workers' health and life.

2. LITERATURE REVIEW

Numerous researchers have explored the development of hardware and software solutions for device control via voice commands. A fundamental aspect of this research has been the creation of various models for recognizing voice instructions.

In 2016, Vajpai and colleagues examined speech signal processing, encompassing automatic speech recognition, synthetic speech, and natural language processing. Their study highlighted the increasing impact of these technologies on business, industry, and the usability of personal computers. They also traced the evolution of speech recognition systems within industrial applications, demonstrating how these advancements facilitate next-generation voice-enabled services. The research provided a comprehensive review of speech recognition technologies, summarizing key insights from existing studies and outlining their applications in sectors such as healthcare, robotics, forensic analysis, defense, and aviation.

The challenge of speech intelligibility in noisy environments, such as hearing aids affected by background noise, was addressed by Park and colleagues in 2016. Their research focused on reducing babble noise without distorting human speech in low signal-to-noise ratio conditions. They proposed a supervised learning approach to map noisy speech spectra to clean speech spectra using fully convolutional neural networks, which require fewer parameters than fully connected networks.

Deng and colleagues (2019) contributed to ensemble learning by developing linear and log-linear stacking techniques for speech-class posterior probability estimation. Their research applied these methods to convolutional, recurrent,

and fully connected deep neural networks, formulating and solving convex optimization problems to enhance the accuracy of phone recognition. The results demonstrated that integrating multiple deep learning models significantly improved hierarchical feature extraction from raw acoustic signals.

Azhiimah and colleagues (2020) reviewed voice-controlled automation using Arduino from 2014 to 2020, analyzing 25 academic journals. Their findings classified voice control into two types: voice recognition, which relies on EasyVR hardware and microcontrollers, and speech recognition, which operates via Android applications. Remote control functionality was achieved through Bluetooth and internet connectivity. The study identified factors influencing system performance, including pronunciation clarity, pitch, microphone distance, sound source, intonation, and noise levels.

Many authors have written on the topic of speech recognition service integrated into an industrial training station. Govoreanu and colleagues (2021) approach leveraged a decentralized microservice architecture, with a speech recognition engine enabling seamless interaction among system components. By incorporating APIs for English and Romanian, the system improved recognition accuracy for task-oriented commands and significantly reduced response times.

In the same year, Abdulkareem and colleagues focused on integrating speech recognition with the Internet of Things (IoT) for home automation. Their model utilized digital signal processing and the hidden Markov model to enhance command accuracy. A cloud-based approach leveraging Google's API enabled internet access for command storage. With 150 recorded speech samples, the system achieved an accuracy rate exceeding 80%.

Recent computing advancements have positioned voice recognition as a biometric technology that enhances security and convenience. However, automatic speech recognition accuracy remains a significant challenge. Fegade and colleagues (2021) reviewed the current state of voice recognition and its industrial applications, particularly in public safety solutions.

In 2022, Rendyansyah and colleagues examined a voice-controlled robotic arm with four degrees of freedom. A computer system managed overall operations, with a single operator providing commands. The recognition model employed Mel-Frequency Cepstral Coefficients and Artificial Neural Networks. Through 90 experimental trials, the system achieved a 94% success rate. However, variations in operator intonation and similar speech patterns occasionally caused errors, and the study did not assess multi-user performance.

Sharma and colleagues (2023) highlighted the advantages of voice recognition in multitasking environments, allowing users to perform manual operations while issuing voice commands. Their study noted the growing adoption of speech recognition in artificial intelligence applications, particularly in voice assistants, smart home devices, and search engines. According to Research and Markets, the global voice recognition market is projected to grow at a compound annual rate of 17.2%, reaching $26.8 billion by 2025.

Hermawanto and colleagues (2024) investigated voice recognition for enhancing door security. Their study employed a trial-and-error approach to develop a system that grants access based on specific voice commands. The findings showed that the system effectively recognized commands within a 10 cm range, with accuracy decreasing as the distance increased beyond 15 cm. The research demonstrated the feasibility of voice-based security solutions but highlighted the need for further refinements in speech reception at greater distances.

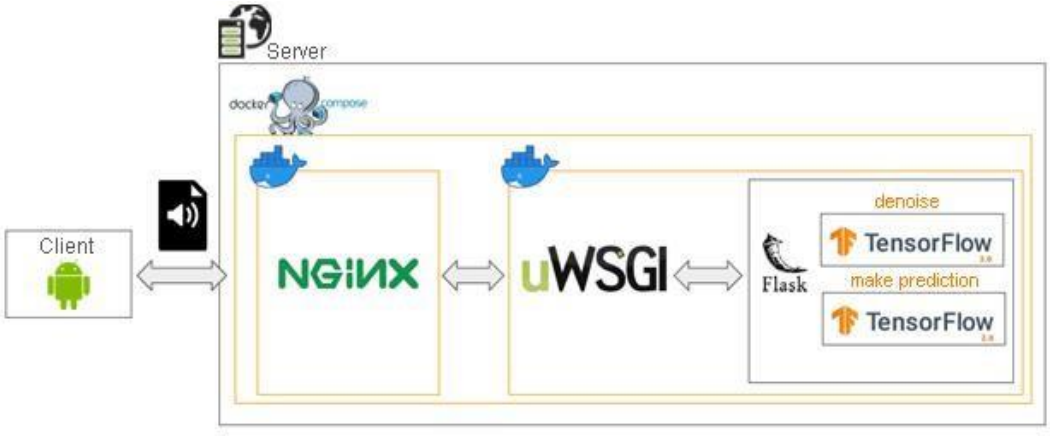
Collectively, these studies illustrate the continuous advancements in voice recognition technology and its diverse applications across industries. While significant progress has been made, challenges remain in terms of accuracy, noise resilience, and multi-user adaptability, pointing to future directions for research and development.

3. DEVELOPMENT OF THE SPEECH RECOGNITION SYSTEM

The architecture of the speech recognition system is shown in Figure 1. The user of this system is a worker in meat processing production. When the worker fills the crates with meat, they issue a voice command to the automated guided vehicle through the voicing system, specifying which chamber and position the pallet with the crates should be delivered to, using a Client (Android) application.

The Client application then sends the user's command (audio file) as a POST request to the Server-side application, where Docker Compose forwards the incoming request to a Docker container running an NGINX Web server. NGINX acts as a proxy and, through Docker Compose, communicates via the uWSGI protocol with another Docker container, forwarding the request through the Web Server Gateway Interface (uWSGI) Unix socket, which essentially represents the application web server.

Figure 1. Architecture of the Distributed Voicing System



Source: Authors

Next, uWSGI sends a callable object to the Python Flask application. The Flask application serves as the endpoint that interacts with TensorFlow neural network models. Upon receiving the audio file, the Flask application first processes it through a neural network for noise reduction to enhance speech clarity. The cleaned audio is then forwarded to a convolutional neural network for prediction and recognition of the spoken command.

After TensorFlow completes the prediction of the audio file, the Flask application returns the recognized word in textual format through the reverse path. First, it is sent to uWSGI, which, through the Docker orchestrator, forwards the response object to the NGINX container that originally initiated the request. NGINX, via Docker Compose, then returns the recognized word in textual format to the Client Android application, which initially sent the POST request to the server.

A key question arises: Why are two web servers, NGINX and uWSGI, necessary?

In short, NGINX is a high-performance web server that provides many essential configurations useful for real-world systems. However, it has very limited integration and support for Python, as it does not natively support the uWSGI protocol, which is required for communication between the Python application and the web server. uWSGI, on the other hand, is an application web server that serves as a gateway, understanding both environments. It translates the incoming POST request to the Python Flask application and later forwards the response object returned by Flask back to the NGINX web server.

The server-side is structured using Docker containers, dividing the system into two main applications. One Docker container is dedicated to the NGINX web server, while the other contains uWSGI, Flask, and TensorFlow. Docker containers isolate independent software packages along with all necessary code and libraries required to run the application.

By leveraging Docker's benefits, a microservices architecture was implemented, enabling cross-platform compatibility, so that containers can run independently on various platforms such as Linux, Windows, and macOS, regardless of the hardware capabilities of the host machine.

To manage the Docker containers, Docker Compose was implemented, allowing the creation of a container network through which the containers can communicate with each other efficiently.

3.1. PREPARATION OF THE SPEECH DATASET

It is necessary to extract the required dataset, which consists of key words that will be used to train the convolutional neural network (CNN). This dataset contains the same set of key words that the trained speech recognition system will later identify. In real-world voicing system implementations, one of the most time-consuming tasks is the collection (recording and processing) of training data. The dataset must contain a large number of samples, specifically audio WAV files, with spoken commands from multiple users who will utilize the system. To save time during the development and presentation of this system, which was built based on the previously described complex architecture, the Google AI dataset was selected. This decision was made due to the limited availability of publicly accessible datasets for sound recognition tasks, as well as the lack of

suitable datasets for keyword recognition. The Google AI dataset was used as the initial base for testing the developed model. It contains approximately 65,000 one-second-long audio files of 30 different spoken commands, recorded by thousands of speakers in real-world environments. The dataset includes commonly used commands such as: "yes", "no", "up", "down", "left", "right", "on", "off", "stop", "go", as well as numbers from 0 to 9 and directional movement commands. This dataset is specifically designed to facilitate the development of basic voice commands for various applications. The first step is to extract features from all audio samples in the dataset and store them in a JSON file. The key features needed for speech recognition are Mel-Frequency Cepstral Coefficients (MFCCs). MFCCs are widely used in speech classification, including speech recognition, musical instrument classification, and music genre detection. The model primarily uses 13 MFCC coefficients, and the extracted data is stored as a two-dimensional array. The first dimension represents the number of time steps (or frames). The second dimension consists of 13 MFCC coefficients for each time step. Essentially, the audio snapshot is divided into segments, with each segment containing 13 MFCC coefficients extracted from the audio waveform. A Python script was developed to process and prepare the data, storing the extracted MFCC features in a JSON file for later use in training the convolutional neural network.

3.2. DEVELOPMENT AND TRAINING OF THE SPEECH RECOGNITION MODEL

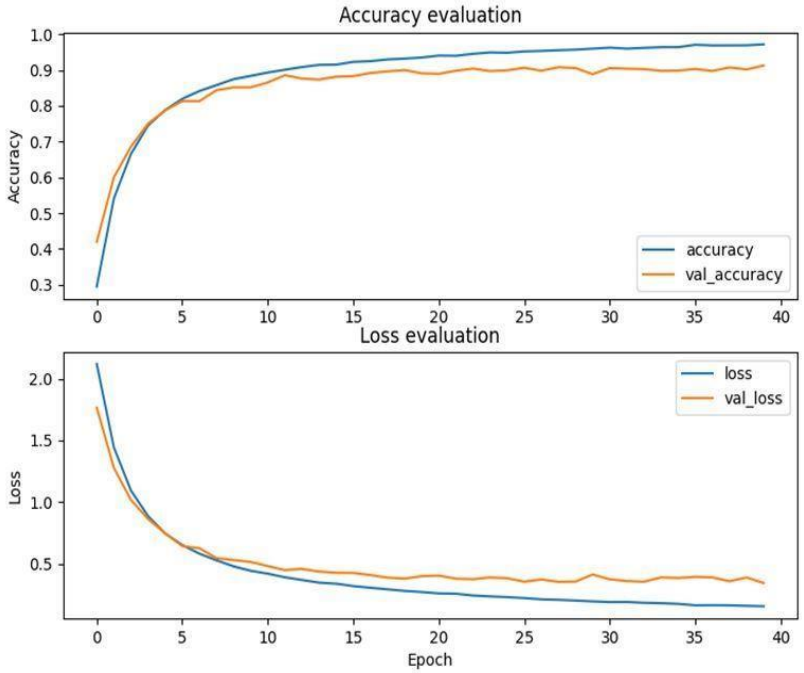
This section describes the development of a speech recognition model using TensorFlow and Keras libraries. A Python script was written to design the architecture of the CNN, compile it, train it, test it, and save the trained model for later use as the core component of the speech prediction system.

The program first loads the dataset for training, validation, and testing. In the next step, the convolutional neural network model is built, then trained, followed by its evaluation using the test dataset, and finally, the trained CNN model is saved. Running the Python script initiates the training process of the neural network, after which the trained model is generated and saved under the name "model.h5".

The number of epochs represents a hyperparameter that defines the total number of times the entire training dataset passes through the neural network during the learning process. The trained algorithm (tested on a dataset of 10 English-language commands) achieved its learning goal after 40 epochs. The obtained loss estimate for this trained model is 0.3477, while its test accuracy for the dataset of 10 words is an impressive 91.25%.

Figure 2 presents the generated graphs from the program, illustrating the validation of losses and model accuracy after each epoch.

Figure 2. Accuracy and Loss Estimation per Epoch



Source: Authors

3.3. PREDICTION USING THE TRAINED SPEECH RECOGNITION MODEL

The next step is to create a keyword recognition service that will load the trained and saved neural network model (model.h5) to perform keyword prediction. For this purpose, a class has been implemented that represents the keyword spotting service, which makes predictions based on the trained neural network model.

To test the trained model, a folder named "test" must be created, containing ".wav" audio files of the words we want to predict (i.e., send to the trained model (model.h5) for recognition). Then, in the main function, within the "predict" method, the file path of the test audio samples must be passed as an argument to perform the prediction.

An example of this Python script is provided below.

if name == " main ":

 # create 2 instances of the keyword spotting service

kss = Keyword_Spotting_Service()

#kss1 = Keyword_Spotting_Service()

 # check that different instances of the keyword spotting service point back to the same object (singleton)

#assert kss is kss1

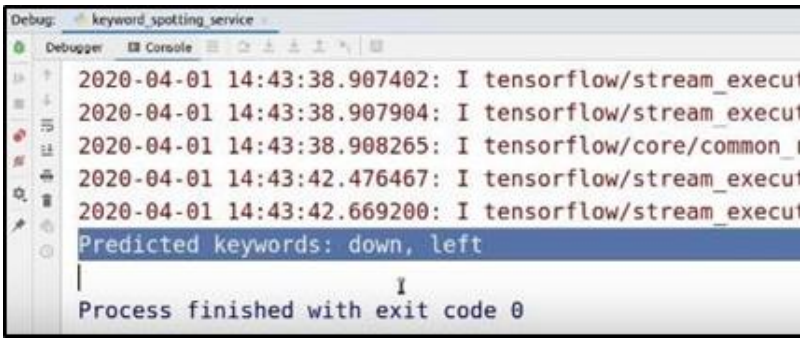
 # make a prediction

keyword = kss.predict("*test/down.wav*") keyword1 = kss.predict("*test/left.wav*")

print(f"Predicted keywords: {keyword}, {keyword1}")

The result of the trained model's accuracy is shown on Figure 3. It can be seen that the trained model performs accurately and successfully predicts the keywords (audio files) it has been trained on.

Figure 3. Prediction of the trained model



Source: Authors

3.4. SPEECH RECOGNITION SYSTEM AS A FLASK API

The following presents the development and application of the TensorFlow neural network model as a Flask API. The speech recognition system (keyword spotting service) shown in the previous section was developed as a Flask application. Then, a client (client.py) was implemented, which can send audio files via HTTP POST requests to the Flask server (server.py) and return predictions. The Flask API on the server calls the keyword_spotting_service for prediction and sends the recognized word (audio file) response back to the client. The process works as follows: the Flask server (server.py) listens on localhost at port 5000 and accepts HTTP POST requests from the client (client.py). The server reads the request, extracts the audio file packaged in the request (POST request), and performs the keyword prediction from the audio file through the keyword_spotting_service. It then returns the prediction made by the trained convolutional neural network model back to the client.

The implemented method provides the following functionalities:

- Accepts and stores the audio file,
- Calls the keyword_spotting_service (wrapper around the trained CNNs model – model.h5),
- Performs the prediction (which word is contained in the incoming audio file),
- Deletes the temporarily stored audio file in the current directory,
- Returns the recognized keyword in JSON format.

In real-world speech recognition systems, Flask cannot be used. Flask is primarily a development server and is not used in production environments. So far, a part of the application has been developed

that sends a POST request with the WAV audio file (keyword), and the Flask development server has been implemented, which accepts the POST request. The Flask application then forwards the audio file to TensorFlow, which analyzes the audio file information, makes a prediction, and then returns the response through the Flask development server to the test script.

The described solution has proven to be a development application, representing the minimal architecture required to create a basic TensorFlow application model, but it cannot serve as a real application. To deploy the application in a real production context, a reliable web server must be included.

3.5. SPEECH RECOGNITION WITH UWSGI WEB SERVER

To achieve a real-world applicable application, it is necessary to configure the uWSGI Web server with the Flask application, which contains the speech recognition system, including the trained convolutional neural network model for keyword recognition.

The architecture has been extended by adding the uWSGI web (HTTP) server between the client application and the Flask application. The client application sends the audio file (POST request), which uWSGI accepts and forwards to the Flask application. Then, Flask calls TensorFlow, which makes the prediction of the keyword based on the trained convolutional neural network model. Afterward, Flask sends the response to uWSGI, which returns the prediction to the client application.

3.6. SPEECH RECOGNITION ON DOCKER PLATFORM WITH NGINX WEB SERVER

Below is the decomposition of the deep learning application for speech recognition using Docker containers. Additionally, the previously presented architecture of the application, specifically the implemented system, will practically include the NGINX web server and the orchestration of Flask and NGINX containers using Docker Compose.

As the first step, an NGINX web server is added in front of uWSGI in the existing architecture, as shown in Figure 4. The reason for adding the NGINX web server is the development of an application based on Docker containers, which ensures its multiplatform capabilities, independent of hardware. This was done to avoid potential configuration issues and to achieve the implementation of a complex architecture at the microservices level.

Figure 4. Application Architecture with NGINX Web Server

Source: Authors

Next, the development of the Docker container network is shown, which contains two containers: one for NGINX and the other for the Flask application that also includes uWSGI, as shown in Figure 5. These containers will be orchestrated by Docker Compose, which will establish a network over which these two containers will communicate. The flow of information after the modified architecture is as follows:

First, an HTTP POST request is sent with the keyword as an audio WAV file from the client. NGINX (which acts as a proxy server) accepts this request and forwards it to uWSGI (using a TCP socket that uses the uWSGI protocol for communication).

Then, uWSGI forwards the data to the Flask application, which invokes the TensorFlow trained model of the convolutional neural network for prediction.

The model predicts the incoming audio file (the keyword), and then the recognized word is returned in text format to the client application.

Figure 5. Containers and Docker

Source: Authors

After installing the Docker platform and configuring Docker Compose, the application was restructured into folders. The files related to data preparation, training, and the JSON file containing useful data about the audio content for training were separated. In the folder named "flask," everything related to the Flask application is stored. This includes the Flask-based speech recognition system, consisting of the keyword_spotting_service script that uses the trained machine learning model for keyword prediction, the saved trained model for speech recognition (model.h5), the Flask server application script (server.py), and the uWSGI configuration file. Everything necessary to contain the Flask Docker container is placed here. In the folder named "nginx," files containing instructions for configuring both Docker containers (NGINX and Flask containers) will be placed.

Then, within the "flask" folder, a Docker file was created, which contains all the necessary instructions and configuration data for Docker to create the container. All of this was done using Python scripts.

3.7. NOISE REMOVAL FOR IMPROVED SYSTEM PREDICTION

Noise represents a significant obstacle for speech recognition systems in terms of their ability and accuracy in making predictions. Noise is also inevitable in production processes and real working environments. If noise is not taken into account during the preparation of the training data for the deep learning model for speech recognition, alongside correct audio features, incorrect and unnecessary features of the audio file, which represent noise, are extracted and sampled. This can later lead to inaccurate predictions from the trained deep learning model. For this reason, it is necessary for the system to automatically remove noise from the keyword before passing the audio file to the trained convolutional neural network (CNN) model for prediction. The audio file will first be passed to an additional trained neural network for noise removal, with the goal of filtering out such noise from the audio file without degrading the signal of interest. This ensures that the CNN model for keyword recognition receives only useful information, i.e., a "cleaned" audio file for prediction. Classic solutions for noise removal in speech typically use generative modeling. In these approaches, statistical methods like Gaussian Mixtures estimate the noise of interest and then recover the signal that contains the noise. However, recent developments have shown that deep learning often surpasses these solutions when noise data is available. Therefore, for the noise removal problem in the practical system, a deep learning model based on CNNs was chosen.

The noisy input signal (audio file with noise) arrives from the client's environment. The goal is to build a statistical model that can extract the clean signal (source) and return it to the user or, in this specific case, forward it to another trained model for prediction. The noise removal model focuses on separating the original speech signal from different types of noise, which the CNN model has been trained on. For the development and testing of the implemented noise removal neural network model to improve predictions, the following publicly available datasets were used:

- Google AI training dataset – used for training the CNN model for prediction,
- The UrbanSound8K dataset.

In audio processing, the neural network needs to extract relevant features from the data. However, before the raw signal is fed into the network, it must be brought into the correct format. First, the audio signal samples (from both datasets) need to be reduced to 8 kHz, and silent frames should be

removed. The goal is to reduce the computation load and the dataset size. All of this audio processing was done using the Python Librosa library. A deep convolutional neural network for speech enhancement was implemented, which is largely based on the scientific paper "A Fully Convolutional Neural Network for Speech Enhancement." A program was created to train the neural network for noise removal. It was trained on the available UrbanSound8K dataset, which consists of 10 types of urban noise. This trained model stopped training and achieved accuracy at the 274th epoch (out of 400 set epochs). Root Mean Square Error RMSE is a useful metric for calculating accuracy. According to the National Digital Elevation Guidelines and FEMA guidelines, the accuracy of the trained model was calculated as 74.63%. In most of the processed examples, the model managed to reduce the noise, but it was not completely eliminated.

Further research could explore the applicability of this algorithm in various production environments. First, it would be necessary to collect a noise dataset from slaughterhouse environments across all production processes. Then, it would be important to test how this noise removal algorithm behaves with the trained model for prediction in Serbian, as well as compare it with other algorithms that serve the same purpose.

3.8. VOICE CONTROL APPLICATION IN SLAUGHTERHOUSES

In order for the speech recognition system to be used in a real environment, two mobile applications were developed. The first application was developed for collecting keywords in the production environment, which are used for training the convolutional neural network model. The second application was developed for slaughterhouse production needs. These mobile Android applications were developed using the C# programming language and the XAMARIN framework. The slaughter process in slaughterhouses begins with the reception of the livestock at the slaughterhouse depot. Upon reception, each animal is tagged with an ear tag, which serves as its unique identifier. The reception of livestock through the voice-enabled slaughterhouse application is shown in Figure 6. The livestock reception process has significantly improved (particularly in terms of accuracy and productivity) with the developed voice control module. This module operates on an Android device, which is attached to the worker's arm. The worker can manually enter the short number from the ear tag, or they can perform this operation through the voice control system. After the ear tag number is recognized, it is sent to the OPIL server, and the recognized number is displayed on the Android device's screen, as shown in the lower-left part of Figure 5. To reduce the amount of data sent to the server for recognition, a Push-To-Talk button (3D printed micro switch) was developed. When the worker presses the micro switch, the microphone function is activated, and the system expects the command to be given. The recognized or manually entered ear tag, as previously mentioned, is recorded through the IoT OPIL platform in the Orion Context Broker (OCB) on the VVT server, where the EarTag entity is created, storing all ear tags in a sequence (readings). When saving the entry, the ERP system is connected to the IoT OPIL Orion Context Broker (OCB) to read data about the ear tags and RFID tags of the half carcasses, in order to store these data for each saved warehouse item.

Figure 6. Voice Control Module Operation in Slaughterhouses

Source: Authors

CONCLUSION

Even fifty years ago, science fiction books were written in which humans controlled various machines with their voices. It was only with the advent of modern computers that this human dream became a reality.

This paper presents the development of a complex architecture for a practical deep learning system, created with the goal of developing a usable speech recognition system. The imperative was to develop a machine learning application, whose model is based on data analysis and artificial intelligence. To this end, convolutional neural network models for prediction and noise removal were developed and presented. These models were developed using the Python programming language and the TensorFlow and Keras libraries, which are tools in the field of data science.

To enable the trained models to be used in the form of an application, an API was developed with a Flask application, through which the trained neural network models for noise removal and prediction are called. A limitation of Flask is that it is a deployment server, which is not used in production environments. To make this application usable in production, Web Server Gateway Interface (WSGI) was implemented in the architecture. WSGI is a software application added to the architecture to develop hosting services, as its native binary protocol allows communication with other servers. WSGI, as a communication gateway, enabled the integration of multiple incompatible technologies and environments into a single system.

NGINX was added as the primary web server to accept HTTP POST requests arriving from the client application. NGINX, through the uWSGI protocol, facilitates communication with the Flask application, which at its core uses TensorFlow and Keras-trained models for deep learning based on the domain of data science. In order to create a system that can be multiplatform, an internal Docker container network was established, managed by the Docker Compose orchestrator. The decomposition of the complex architecture into separate containers led to the realization of the existing architecture at the microservice level. This allows for any changes to the neural network model or any other separate container without affecting the system's overall operation. Additionally, it enables easy changes to the architecture and parallel development of the application by multiple teams, where each team can focus on developing a separate container, representing an independent unit.

The client Android application, as well as the module for generating keywords for training, presented in this paper, enabled the practical application of this system in real-world working environments and demonstrated its benefits. Some of the advantages brought by the presented system include:

- Overall production efficiency.
- Significantly higher labor productivity and thus profitability.
- Improvement of data traceability in complex production.
- The possibility of implementation and integration into robotics and logistics.

The next step in upgrading this work is conducting research on real data, which the presented architecture allows. It is necessary to investigate the accuracy of predictions made by the realized model in real production conditions, using a collected training data set in Serbian. Then, the accuracy of the prediction of the existing algorithm on the collected data should be evaluated, and the results should be compared with those obtained using other available algorithms and deep learning models. Research should also be done on the effectiveness of prediction in real environments, both with and without the application of the deep learning model for noise removal. A comparison of multiple noise removal algorithms should be conducted. The behavior of the trained models and the system should be monitored at different production locations (different machinery, noise levels, and production processes).

The adaptation of the model for individual users, with and without noise removal algorithms, should also be explored. Training and testing on a data set from a single speaker compared to training on a dataset containing recordings from multiple speakers of different ages and genders should be done. The accuracy of models trained on male and female speakers needs to be verified. Further research should be conducted in terms of monitoring productivity, efficiency, and effectiveness, comparing the use of this system to existing solutions and the previous production process.

We hope that this proposed model will be applicable to various manufacturing companies.

ACKNOWLEDGEMENTS

This research was supported by the Ministry of Scientific and Technological Development and Higher Education of the Republika Srpska under the Agreement on Co-financing of the Scientific and Research Project No.: 19.032/961-47/24 dated 30.12.2024.

REFERENCES

[1] Balamurugan, S., Ayyasamy, A. & Joseph, K.S. (2021). IoT-Blockchain driven traceability techniques for improved safety measures in food supply chain. Int. J. Inf. Technol, 1–12.

[2] Damjanović, S. & Popović, B. (2008). Praćenje proizvoda RFID tehnologijom. Novi Ekonomist, broj 4, 25 - 28.

[3] Damjanović, S., Katanić, P. & Drakul, B. (2021). The impact of the covid-19 pandemic on the global community's mobility. Novi Ekonomist, Vol 15(2), broj 30, 15-23.

[4] Fearne, A. (1998). The evolution of partnerships in the meat supply chain: Insights from the British beef industry. Supply Chain Management, 3(4), 214–231.

[5] Hobbs, J.E. (2021). The Covid-19 pandemic and meat supply chains. Meat Science, Volume 181, 1-6.

[6] Ilić, S., Damjanović, S. & Katanić, P. (2023). Prednosti i nedostaci primjene pametnih kućanskih uređaja. Zbornik radova EKONBIZ 2023 (129-143). Bijeljina: University of East Sarajevo, Faculty of Business Economics Bijeljina.

[7] Katanić, P. & Damjanović, S. (2021). Otkrivanja prevare prilikom prodaje proizvoda preko interneta. Zbornik radova EKONBIZ 2022 (152-161). Bijeljina: University of East Sarajevo, Faculty of Business Economics Bijeljina.

[8] Kumar, V. (2016). New kid on the blockchain. Focus Blockchain, Vol. 8 No. 3, 19-22.

[9] Liu, P., Hendalianpour, A., Hamzehlou, M., Feylizadeh, M. & Razmi, J.. (2021). Identify and rank the challenges of implementing sustainable supply chain blockchain technology using the bayesian best worst method. Technol. Econ. Dev. Econ. 27, 656–680.

[10] Mai, N., Bogason, S.G., Arason, S., Árnason, S.V. & Matthíasson, T.G. (2010). Benefits of traceability in fish supply chains – case studies. British Food Journal, Vol. 112 No. 9, 976-1002.

[11] Pal, A., & Kant, K. (2019). Using Blockchain for Provenance and Traceability in Internet of Things-Integrated Food Logistics. Computer, 52(12), 94–98.

[12] Steven, A. B. (2015). Supply Chain Structure, Product Recalls, and Firm Performance: Empirically Investigating Recall Drivers and Recall Financial Performance Relationships. Decision Sciences, 46(2), 477–483.

[13] Wang, S., Ghadge, A., Aktas, E. (2024). Digital Transformation in Food Supply Chains: An Implementation Framework. Supply Chain Management, Vol. 29 No. 2, 328-350.

[14] Xu, J., Guo, S., Xie, D. & Yan Y. (2020). Blockchain: A new safeguard for agri-foods. Artificial intelligence in agriculture 4(1), 1-32.

[15] Yadava, V.S., Singh, A.R., Raut, R.D., Mangla, S.K., Luthr, S. & Kumar A. (2022). Exploring the application of Industry 4.0 technologies in the agricultural food supply chain, A systematic literature review. Computers & Industrial Engineering 169(S1). ISSN 0360-8352.

This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License