Speaker Recognition using CNN

Hetansh Shah
Electronics and Communication
Institute of Technology, Nirma University
Ahmedabad, Gujarat
21bec045@nirmauni.ac.in

Himangi Agrawal
Electronics and Communication
Institute of Technology, Nirma University
Ahmedabad, Gujarat
21bec046@nirmauni.ac.in

***Abstract*—The process of automatically recognizing or identifying a person based on the features of their voice is known as speaker recognition. A speaker profile is created by taking specific features from a speaker's speech signal and analyzing them. This profile can then be used to identify or confirm the speaker's identity in future interactions. Speaker recognition is a crucial component in various applications, including security systems and personalized services. In this paper, a speaker recognition system utilizing Convolutional Neural Networks (CNNs) is trained on audio spectrogram data. The model is trained with categorical cross-entropy loss using the Adam optimizer, and accuracy metrics are used to assess the model's performance. Precision-recall curves are also used to evaluate how well the model performs, especially when dealing with unbalanced datasets. The model is trained for pwm speeches dataset and achieves an accuracy of 99.2% . The outcomes show how well the suggested CNN-based method works for speaker recognition tasks and point to its potential for practical uses in biometric authentication and audio processing.**

***Index Terms*—Speaker recognition, Convolutional Neural Networks, datasets, spectrogram, adam optimiser, neural networks, model accuracy.**

I. Introduction

Speaker recognition, a subset of biometric identification systems, plays a pivotal role in various domains where the authentication or identification of individuals is paramount. The ability to discern and authenticate individuals based on their unique vocal characteristics holds immense potential in security systems, telecommunications, and forensic investigations. Unlike traditional authentication methods such as passwords or PINs, speaker recognition offers a non-intrusive and natural means of identification, making it particularly appealing for applications where user convenience is paramount. The majority of the research relies on the user speaking a keyword to trigger the identity-inputting device[1]. Text-dependent speaker identification operates under the assumption that certain aspects of a speaker's speech are reliable and consistent, especially when the speaker is uttering a particular passphrase or phrase. Their distinct vocal patterns, pronunciation, intonation, and other aspects of their speech are examples of these traits. The reference for locating or confirming the speaker is a passphrase or particular phrase. The speaker and the system are both aware of this pre-chosen passphrase. The speech signal of the speaker's passphrase is analyzed by the system, which then compares it to the reference model of the same passphrase that has been stored and linked to the claimed identity. The premise behind text-independent speaker identification is that every person has distinct vocal traits that can be recorded and utilized to identify them. Text-independent systems are able to recognize speakers from any spoken content, while text-dependent systems are dependent on certain phrases. Each person has unique vocal characteristics that are shaped by a variety of factors, including physiological variations, speech habits, pronunciation patterns, and the size, shape, and tension of the vocal tract. These qualities come through in speaking rate, intonation, pitch, and timbre. The speech signal is processed to extract pertinent features that capture the distinctive qualities of the speaker's voice in text-independent speaker identification. Mel-frequency cepstral coefficients (MFCCs), linear predictive coding (LPC) coefficients, energy contours, fundamental frequency (pitch), and spectral characteristics are examples of features that are frequently used. After the features are extracted, a speaker model is made to reflect each person's distinct voice traits. In subsequent interactions, this model can be used as a reference to identify or confirm the speaker. For speaker modeling, a variety of statistical modeling methods can be applied, including Gaussian Mixture Models (GMMs), Hidden Markov Models (HMMs), and contemporary deep learning techniques. In the identification process, the system's database's stored speaker models are compared with the features that are extracted from an unknown speech sample. Based on a comparison of the likelihood measure or similarity score between each speaker model and the input features, the system decides which one to use. However, CNN-based methods provide a data-driven paradigm for speaker identification; using multiple layers of convolutional and pooling operations, the network learns hierarchical representations of speech features by taking advantage of CNNs' hierarchical structure, these models are able to automatically extract and encode pertinent patterns and characteristics from the audio data that is being input, which improves performance in speaker identification tasks.

In this paper, we have utilised Convolutional Neural Network (CNN) for speaker recognition from audio spectrogram data. Adam optimizer is used to train the model, and validation data is used to track training progress. Testing data are used to gauge the model's performance, and a visualization of the training history is used to look for signs of over-fitting and convergence.

II. Literature review

There are several methods for resolving the speaker recognition issue. The two parameters that define the solution are the feature of the voice signal to be used, such as vector quantization, artificial neural networks (ANN), Gaussian mixture model (GMM)[2], Mel Frequency Cepstral Coefficients (MFCC), Linear Prediction Cepstral Coefficient (LPCC), and Perceptual Linear Prediction (PLP), and the modeling technique used to learn the voice samples[1]. In order to classify objects of interest and identify speakers based on matching scores,[2] Aroon, and Dhonde used speaker modeling techniques such as Gaussian Mixture Modeling (GMM). Mel frequency Cepstrum coefficients are extracted through a series of steps that include framing, windowing, the Fast Fourier Transform (FFT), and Mel spectrum conversion. Reynolds proposed high performance speaker identification and verification systems in [3] that were based on Gaussian mixture speaker models. The following publicly accessible databases were used to evaluate the systems: TIMIT, NTIMIT, Switchboard, and YOHO. Endres, Bambach and Flosser suggested using spectrograms to analyze audio data in [4]. it suggested that a useful tool for researching age-related changes, voice disguise, and voice imitation is the analysis of voice characteristics using spectrograms. This method offers insights into the stability and variability of voice features over time and across different individuals. The authors of [5] employed maximum likelihood linear regression to create speaker adaptation models using Hidden Markov models (HMMs). Using adaptation data from a new speaker, the MLLR approach updates the model mean parameters to maximize the likelihood of the adaptation data. Assuming that speakers are defined by their means, this approach focuses on their primary distinctions[5]. The NIST dataset served as the basis for the work in [6], where the authors used Support Vector Machines (SVMs) to carry out speaker verification tasks. SVM with support vectors is used by the speaker verification system to make accept/reject decisions based on output scores in relation to a threshold. Training entails creating weighted probability vectors, training with a one-versus-all approach, and treating conversation sides as documents.

Using a specially created dataset that includes audio samples from different speakers in terms of gender, language, and accents, the study by [7] S. Ganvir and N. Lal focuses on speaker verification. Two different kinds of data sets were produced by processing the dataset. The extracted spectral features, such as the spectral bandwidth, MFCCs, and spectral centroid, were used to train an artificial neural network (ANN). The features from the spectrogram images were extracted using another network that had hidden convolutional layers. Audio segments were transformed into spectrogram images for the CNN data set, yielding 19,000 training samples and 8,200 validation samples. Convolutional neural networks, also referred to as "CNNs," have been used in speaker recognition, computer vision, and natural language processing [7] for feature extraction. The CIFAR-10 architecture was altered to create a CNN, which was then trained using spectrograms from 80 distinct speakers. Spectrograms are graphs that show the strength of an audio signal over time. They are composed of various frequencies with a particular waveform. A four-layer architecture characterizes the CNN model presented in [7]. The first two layers have 32 filters each, while the next two layers have 64 filters each, all with a 3x3 kernel size. Max pooling is used between the two sets of layers at a factor of two. Softmax activation function is used prior to the last dense layer, and ReLU activation function is used after each layer. The fourth convolutional layer's output is flattened into a representational vector for transfer learning. The model is trained with the Adam optimizer, and regularization is ensured by applying a 25 percent dropout to the convoluted layers. The ANN approach was contrasted with the CNN model. About 90 percent of the CNN model's accuracy was achieved[7] for 100 epochs. It was determined that the CNN model, which can extract features from a larger number of speakers, is the best overall. However, training on a large training data set takes longer. An ANN's model accuracy was found to be 93 percent, but it functions well enough when there are fewer speakers and less training data is needed to fully train the model.

The author employs a Preprocessing block in [1]. The voice samples are preprocessed in this block in order to reduce or filter the noise content and remove any silent portions from the signal. The Feature Extraction block comes next and is in charge of extracting MFCCs. After that, it passes these vectors to the speaker modeling block after reducing their dimension. then MFCCs are the input used by the speaker modeling block to create a model. then, only the recognition phase contains the Computing Likelihood block. In order to identify the speaker, it looks for a match in the trained model and gives it back as output. The CNN model [1] was used to test the pre-processed and extracted features. The input vector's dimensions matched the features' sizes in a single frame. There were 52 features per frame, including 13 MFCC. There were two learning models used: CNN and DNN. Three convolutional layers and a dense layer at the end made up the CNN's four layers. 52 convolution kernels with filter lengths (window sizes) of 13 and 7, respectively, are used in the first two layers. With a window size of three, the third convolution layer employs thirteen output kernels. After flattening the output, a dense layer with 1000 nodes is created. At this point, 0.25 dropout is added to prevent overfitting. The last output layer comes next with adam [1] as an optimizer. Two different test sets were taken, as shown in Fig [1] and Fig [2].

For real-world samples [1], up to 10 speakers can be fed into the network with a respectable accuracy of 75–80 percent (Fig [2]). The accuracy of the model for voice recorded is 70 percent(Fig[2]). For five speakers, DNN provides an accuracy of 75–80 percent (Fig [1]) and after that, its performance deteriorates. In conclusion, the CNN model in [1] produced an accuracy of 75–80 percent (Fig[2]).

No. of Speakers	**DNN accuracy**	**CNN accuracy**
5	70	76
10	67	68
15	70	72
20	55	58
35	55	58
50	61	71

Fig. 1. Speaker recognition dataset from openslr[1]

5	80	90
6	65	78
7	64	77
8	58	75

Fig. 2. Observations for Real-World Voice Samples[1]

III. PROPOSED WORK

Our work on the task of speaker classification using machine learning, employes a Convolutional Neural Network (CNN) architecture specifically designed for processing audio data. Fig[3] shows the high-level architecture of model.

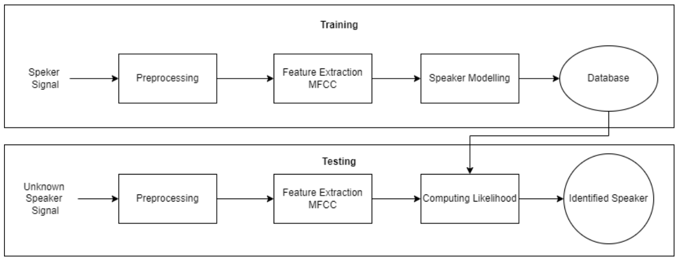


Fig. 3. Flowchart

The audio data is ready for training and testing the Convolutional Neural Network (CNN) model for speaker recognition thanks to the preprocessing block. It uses the librosa library to load each audio file, after which it extracts the features of the Mel-scaled spectrogram. To increase the dynamic range, the frequency content of the audio signal is represented over time using a melan-spectrogram and then converted to decibels (dB). The preprocessing block creates spectrogram images from raw audio data and gets them ready for training and testing the CNN model with matching labels. It guarantees that the model receives consistent and appropriately formatted data for efficient learning and classification of speaker identities by standardizing the input data and encoding the labels. The feature extraction block is in charge of MFCC extraction. After that, it passes these vectors' reduced dimensions to the speaker modeling block. Speaker modeling block creates a model using MFCCs as input.

A. *DATASET AND FEATURE EXTRACTION*

The dataset is taken from [8] which contains 5 folders for speech samples of 5 different speakers. Each folder contains 1500 audio files, each 1 second long and sampled at 16000 Hz. Spectrophotoms, which are graphic depictions of the frequency content of an audio signal over time, are created from audio files. The neural network can now analyze the audio data in a format that captures frequency and temporal information, both of which are essential for speaker classification tasks. Eighty percent of the spectrogram images are set aside for training, and the remaining twenty percent are set aside for testing. This guarantees that the model's performance is assessed using hypothetical data, resulting in an accurate assessment of the model's capacity for generalization. To ensure reproducibility, a random state of 42 is added during the splitting procedure. The librosa library is primarily used for feature extraction, which involves computing Mel-scaled spectrogram features from audio files. The audio signal is split up into brief overlapping frames, and each frame is then subjected to a Fourier transform to produce the Mel-scaled spectrogram. The magnitude of the frequencies in each frame is represented by the resulting spectrogram. The frequency axis receives the application of the Mel scale, which is more in line with human auditory perception. The lower frequencies are highlighted on this scale because they are more important for speech analysis. The Mel-scaled spectrogram is converted to decibels (dB) after computation. The spectrogram's dynamic range is increased by this conversion, improving its suitability for analysis and visualization. The resulting Mel-scaled spectrogram, which captures crucial characteristics for speaker recognition tasks, shows the frequency content of the audio signal over time in decibels. The Convolutional Neural Network (CNN) model learns discriminative patterns for speaker recognition by using these features as input for training and classification.

B. *CNN MODEL*

Three convolutional layers precede an activation function known as a Rectified Linear Unit (ReLU) in each of the model's subsequent layers. These layers are in charge of identifying features and spatial patterns in the spectrogram pictures. The first convolutional layer has a kernel size of (3, 3) and 64 filters. A kernel size of (3, 3) is used in 32 filters of the second convolutional layer. There are 16 filters in the third convolutional layer, each with a kernel size of (3, 3). A Max Pooling layer is added following each convolutional layer. The feature maps that are derived from the convolutional layers are downsampled by this layer. When working with 2D input data, the Max Pooling layer usually uses feature maps that are acquired from convolutional layers. It preserves significant features while reducing the input data's spatial dimensions. The size of the pooling window that is used for downsampling is specified by the pool size parameter. It is set to (2, 2) in this instance, denoting a 2x2 window. The largest value in each window is chosen during the pooling process, which essentially reduces the size of the feature maps by a factor of two in both the height and width dimensions. MaxPooling

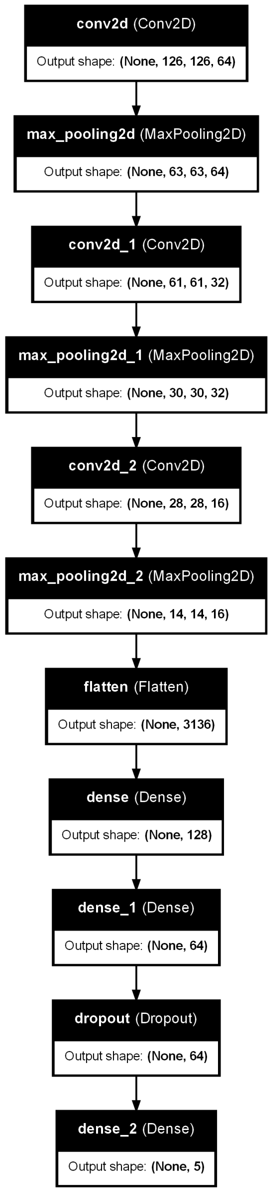

conv2d (Conv2D)
Output shape: (None, 126, 126, 64)
max_pooling2d (MaxPooling2D)
Output shape: (None, 63, 63, 64)
conv2d_1 (Conv2D)
Output shape: (None, 61, 61, 32)
max_pooling2d_1 (MaxPooling2D)
Output shape: (None, 30, 30, 32)
conv2d_2 (Conv2D)
Output shape: (None, 28, 28, 16)
max_pooling2d_2 (MaxPooling2D)
Output shape: (None, 14, 14, 16)
flatten (Flatten)
Output shape: (None, 3136)
dense (Dense)
Output shape: (None, 128)
dense_1 (Dense)
Output shape: (None, 64)
dropout (Dropout)
Output shape: (None, 64)
dense_2 (Dense)
Output shape: (None, 5)

Fig. 4. CNN Flowchart

layers lower the computational complexity of the model by downsampling the feature maps. They accomplish this by making the model more efficient by lowering the quantity of parameters and calculations needed in later layers. Furthermore, MaxPooling layers aid in keeping important features while removing irrelevant data. They achieve this by maintaining key spatial patterns by holding onto the most significant activations within each pooling window. The model architecture includes a Flatten layer in addition to the convolutional and pooling layers. The function of the flatten layer is to create a one-dimensional vector from the multi-dimensional feature maps that the convolutional and pooling layers produced. A distinct feature or pattern that the convolutional layers have learned is represented by each feature map. These feature maps can be flattened so they can be concatenated into a single vector, which is then used as the input for fully connected layers. By concatenating the feature map rows, the Flatten layer effectively reshapes the 2D feature maps into a 1D vector. The Flatten layer will reshape the output of the final Max Pooling layer into a vector of size (N * M * C), for instance, if the output has dimensions (N, M, C), where N and M are the spatial dimensions of the feature map and C is the number of channels or filters. As the data is being prepared for input into the fully connected layers, this reshaping process maintains the spatial relationships that the convolutional layers have learned. Following flattening, the fully connected layers (Dense layers) input is the one-dimensional vector that was extracted from the Flatten layer. Based on the features that the convolutional layers extracted, fully connected layers carry out the classification task. Between the convolutional layers, which extract spatial features, and the fully connected layers, which carry out classification based on these features, the flatten layer serves as a link.

Two fully connected layers are added to the model after the Flatten layer. Because every neuron in a fully connected layer is connected to every other neuron in the layer above it, fully connected layers are also referred to as dense layers. There are 128 neurons in the first layer that is fully connected. Every neuron in the Flatten layer's output is connected to every other neuron in this layer. ReLU is the activation function that is employed in this layer. Rectified linear units, or ReLUs, are a popular option for neural network activation functions because they add non-linearity to the model. 64 neurons make up the second fully connected layer. Every neuron in this layer is connected to every other neuron in the first fully connected layer's output, just like in the first layer. This layer also uses the ReLU activation function. ReLU adds non-linearity to the model, enabling the network to decipher intricate relationships and patterns in the data. Based on the size of the input data and the complexity of the task, the number of neurons in each fully connected layer—128 in the first layer and 64 in the second—was chosen during the design process. These fully connected layers use the features that the convolutional layers extracted to carry out the classification task. Using the learned parameters (weights and biases), they learn to map the high-level features from the flattened input to the output classes (speakers). To sum up, they help the model to effectively categorize speakers according to the characteristics that are taken out of the audio data's spectrogram representations. A regularization method called dropout keeps neural networks from overfitting. In order to essentially disable input units (neurons), it drops a certain percentage of them at random during training. Layers that are fully connected are added before the dropout layer (Dropout(0.5)). The dropout rate—a measure of the percentage of input units that will be dropped during training—is indicated by the parameter 0.5. Here, fifty percent of the input units are dropped at random.

To ensure that different sets of neurons are dropped in each training iteration, the Dropout layer randomly chooses which units to drop with a probability of 0.5. Dropout keeps the model from depending too much on any one subset of neurons by randomly dropping input units. As a result, the network performs better on untested data since it is forced to learn more reliable and generalizable features. In order to lower the risk of overfitting, dropout functions as a type of ensemble learning in which several sub-networks are trained using various subsets of neurons removed. The primary objective of incorporating Dropout is to improve the model's capacity for generalization. Dropout helps the model learn more abstract and generalizable representations by keeping it from memorizing the training data too thoroughly. As a result, the model performs better overall and is more robust when applied to unknown data.

The output layer, which is the final layer in the CNN model architecture, is a fully connected layer. The quantity of speaker classes and neurons in this layer are the same. In this layer, every neuron is associated with a particular speaker class. The softmax activation function is the one utilized in the output layer. Softmax is frequently utilized in multi-class classification tasks because it turns the neurons' raw output scores into probabilities. It is appropriate for representing class probabilities because it guarantees that the output probabilities add up to 1. The softmax function uses the output scores of the neurons in the output layer to calculate the probability distribution over the various speaker classes. The output layer's neurons each generate a score that indicates how confident the model is in its ability to predict the associated speaker class. The softmax function is then used to convert these scores into probabilities. The output probabilities show how likely it is that a given input will be correctly classified for each speaker class. The class that the softmax function predicts will have the highest probability output is chosen as the predicted speaker class during inference or prediction. By offering class probabilities, the softmax activation function helps users evaluate the model's confidence in its predictions and makes it easier to interpret the model's predictions.

During training, the model parameters are updated using the Adam optimizer. One popular option for training neural networks, including CNNs, is the Adam optimizer. It is a development of stochastic gradient descent (SGD) that combines momentum-based optimization and adaptive learning rate techniques. Adam modifies the learning rate according to the gradients' first and second moments for every parameter. In comparison to traditional SGD, it aids in faster and more reliable convergence, particularly for problems involving large datasets or high-dimensional parameter spaces. When dealing with multi-class classification problems where the output labels are one-hot encoded, the categorical cross-entropy loss function is frequently utilized. It calculates the discrepancy between the true probability distribution (ground truth labels) and the predicted probability distribution (softmax layer output). When the model predicts the incorrect class with high confidence, it is penalized more severely. In the compile() function, categorical cross entropy is defined as the loss function. This suggests that the categorical cross-entropy between the true and predicted distributions will be minimized by the model by optimizing its parameters. The adaptive learning rate and momentum features of the Adam optimizer facilitate effective navigation of the model's parameter space, resulting in improved optimization and faster convergence. The model's performance is measured by the categorical cross-entropy loss function, which makes sure the model can effectively learn to distinguish between various speaker classes.

Five epochs are used to train the model. By expanding the number of epochs, the model can potentially see better results and see more examples. But it's critical to keep an eye out for overfitting, which occurs when the model becomes so adept at memorizing training data that it struggles to generalize to new data. The number of samples processed prior to the model's parameters being updated (i.e., prior to a weight update step) is determined by the batch size. For training, a batch size of 32 has been set. To evaluate the model's capacity for generalization, it is crucial to track how well it performs on data that it has never seen before during training. A validation split of 20% is applied in the code that is provided. This indicates that 80% of the training data will be used for training and 20% will be set aside for validation. Instead of being used for training, the validation data is used to assess the model's performance at the end of each epoch. This makes it possible to check for overfitting and aids in hyperparameter tuning. When verbose=1 is used, the training progress—including the accuracy and loss metrics for each epoch—is shown in the console.

C. Pseudo Code

- Spectrograms: $X_{\text{train}}, X_{\text{test}}$ (converted from audio files)
- Labels: $Y_{\text{train}}, Y_{\text{test}}$ (speaker names)

Data Splitting

- 80% training, 20% testing (random state $= 42$)

CNN Architecture

- Model: Sequential
- Layers:

Model Compilation

  - Optimizer: 'adam'
  - Loss: 'categorical_crossentropy'

Training and Validation

  - Validation Split: 20%
  - Batch Size: 32

Mathematical Representation

Convolution Operation

$$\text{output} = \text{ReLU}(W * \text{input} + b)$$

MaxPooling Operation

$$\text{output} = \text{MaxPooling}(2 \times 2)(\text{input})$$

Dense Layer

$$\text{output} = \text{ReLU}(W \times \text{input} + b)$$

Softmax Activation

$$\text{output}i = \frac{e^{z_i}}{\sum j = 1^N \ e^{z_j}}$$

Model Compilation

- Optimizer: 'adam'
- Loss: 'categorical_crossentropy'

Training and Validation

- Validation Split: 20%
- Batch Size: 32

Mathematical Representation:

Convolution Operation

$$\text{output} = \text{ReLU}(W * \text{input} + b)$$

MaxPooling Operation

$$\text{output} = \text{MaxPooling}(2 \times 2)(\text{input})$$

Dense Layer

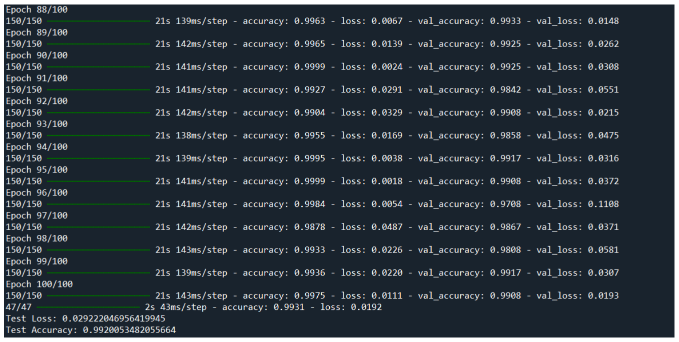
$$\text{output} = \text{ReLU}(W \times \text{input} + b)$$

Softmax Activation

$$\text{output}_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}}$$

IV. EXPERIMENTS AND RESULTS

The model is first trained through 100 epochs. Next, we learned about validation accuracy, validation loss, model accuracy, and model accuracy as shown in Fig[5].



```
Epoch 88/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 139ms/step - accuracy: 0.9963 - loss: 0.0067 - val_accuracy: 0.9933 - val_loss: 0.0148
Epoch 89/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 142ms/step - accuracy: 0.9965 - loss: 0.0139 - val_accuracy: 0.9925 - val_loss: 0.0262
Epoch 90/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 141ms/step - accuracy: 0.9999 - loss: 0.0024 - val_accuracy: 0.9925 - val_loss: 0.0308
Epoch 91/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 141ms/step - accuracy: 0.9927 - loss: 0.0291 - val_accuracy: 0.9842 - val_loss: 0.0551
Epoch 92/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 142ms/step - accuracy: 0.9904 - loss: 0.0329 - val_accuracy: 0.9908 - val_loss: 0.0215
Epoch 93/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 138ms/step - accuracy: 0.9955 - loss: 0.0169 - val_accuracy: 0.9858 - val_loss: 0.0475
Epoch 94/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 139ms/step - accuracy: 0.9995 - loss: 0.0038 - val_accuracy: 0.9917 - val_loss: 0.0316
Epoch 95/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 141ms/step - accuracy: 0.9999 - loss: 0.0018 - val_accuracy: 0.9908 - val_loss: 0.0372
Epoch 96/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 141ms/step - accuracy: 0.9984 - loss: 0.0054 - val_accuracy: 0.9708 - val_loss: 0.1108
Epoch 97/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 142ms/step - accuracy: 0.9878 - loss: 0.0487 - val_accuracy: 0.9867 - val_loss: 0.0371
Epoch 98/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 143ms/step - accuracy: 0.9933 - loss: 0.0226 - val_accuracy: 0.9808 - val_loss: 0.0581
Epoch 99/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 139ms/step - accuracy: 0.9936 - loss: 0.0220 - val_accuracy: 0.9917 - val_loss: 0.0307
Epoch 100/100
150/150 ━━━━━━━━━━━━━━━━━━━━ 21s 143ms/step - accuracy: 0.9975 - loss: 0.0111 - val_accuracy: 0.9908 - val_loss: 0.0193
47/47 ━━━━━━━━━━━━━━━━━━━━ 2s 43ms/step - accuracy: 0.9931 - loss: 0.0192
Test Loss: 0.029222046956419945
Test Accuracy: 0.9920053482055664
```

Fig. 5. Model trained for 100 epochs

A. Model Accuracy

The percentage of correctly classified instances (or samples) in the dataset relative to all instances is known as the model accuracy. It is a performance metric that measures the accuracy with which the model can predict the future. The accuracy attained on the validation set during training is commonly referred to as the model accuracy, which is calculated during the training process. Following a predetermined number of epochs for model training, the accuracy metric displays the percentage of correctly classified samples within the validation set. The model accuracy trend over the course of training is depicted in the Fig 6. A rising curve shows that throughout the course of subsequent epochs, the model is learning and becoming more adept at making accurate predictions. On the other hand, variations or a plateau in the curve can indicate difficulties or constraints encountered throughout the learning process.

B. Model Loss

The value of the loss function throughout each epoch is represented by the model loss curve in Fig 6. Model loss is a metric that expresses how well the model's predictions match the actual (ground truth) values in the dataset. It is also referred to as the loss function or cost function. For every sample in the dataset, it measures the discrepancy between the true and expected output. As part of model optimization, the model loss is calculated during training. Minimizing this loss function, or more specifically, the difference between the model's predictions and the actual labels, is the aim of training. Since the problem involves multiple classes, the categorical cross-entropy loss function is used when compiling the model. The model is minimizing mistakes and convergent toward ideal performance when the loss curve is lowering. On the other hand, irregularities or spikes in the curve may point to overfitting, underfitting, or other problems that need to be addressed.

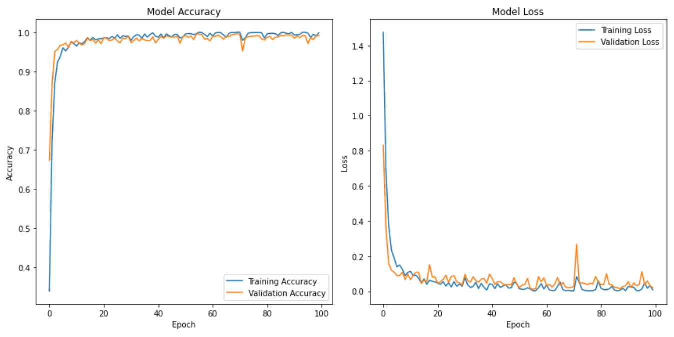


Fig. 6. Model Accuracy and Model Loss for 100 epochs

C. Precision - Recall curve

The spectrogram of input audio is shown in Fig 8. A graphical tool for assessing a classification model's performance is the precision-recall (PR) curve, which is especially useful in situations where there is an imbalance in the distribution

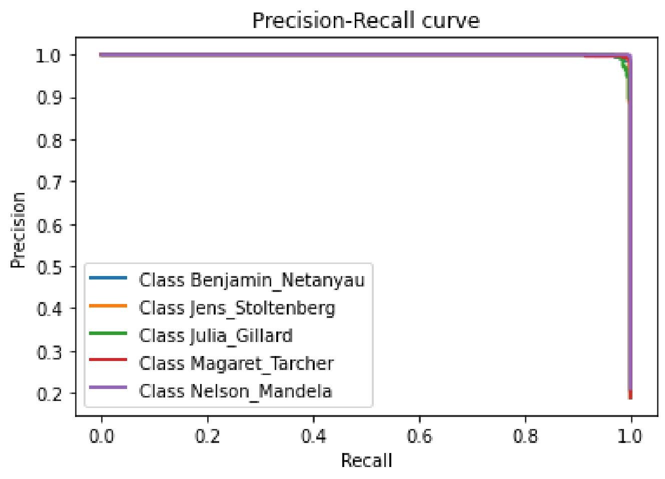


Fig. 7. Precision Recall curve for 100 epochs

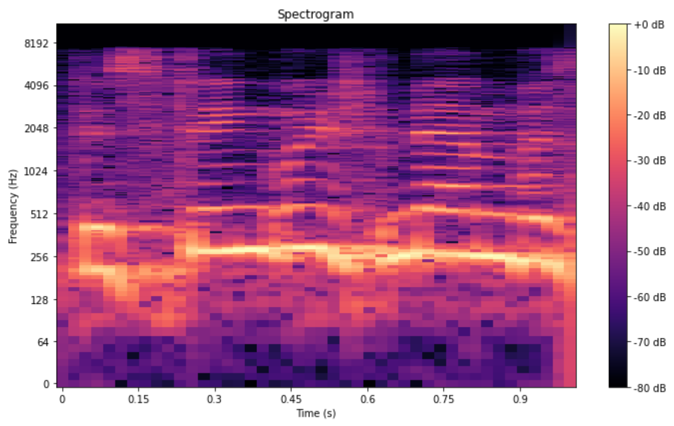


Fig. 8. Spectrogram of input Audio

of classes. Plotting the precision versus recall trade-off for various thresholds placed on the predicted probabilities is done.

The precision-recall curve , figure 7 sheds light on how recall and accuracy are traded off for various classification job thresholds. Precision highlights the model's ability to prevent false positives by measuring the percentage of real positive predictions among all positive predictions. On the other hand, recall highlights the model's capacity to catch pertinent events by calculating the percentage of true positives accurately detected out of all actual positives. This trade-off is represented

TABLE I
COMPARISON FOR DIFFERENT EPOCHS

Epochs	***Test Accuracy***	***Test Loss***
5	97.73	7.73
10	98.00	6.32
20	98.40	4.92
50	99.20	3.96
100	99.20	2.92

graphically by the precision-recall curve, where the optimal curve hugs the upper-right corner and simultaneously shows high recall and high precision.

The test accuracy and test loss are recorded for different number of epochs as shown in Table 1.

It is evident that for 100 epochs with 5 speakers, the model yields the highest accuracy of 99.20%. It can be compared with Fig. 1[1] which results in CNN accuracy of 76%. Our model have acheived accuracy ranging from 97.73% for 5 epochs to 99.2% for 100 epochs.

D. Predicted Output

The final predicted is the final output of this model. As seen in the Figure 9, our model produces the expected output with the associated probability. The saved model "speaker-recognition-model.h5" is used to predict the final output and shown in the bar graph are their predicted probabilities.

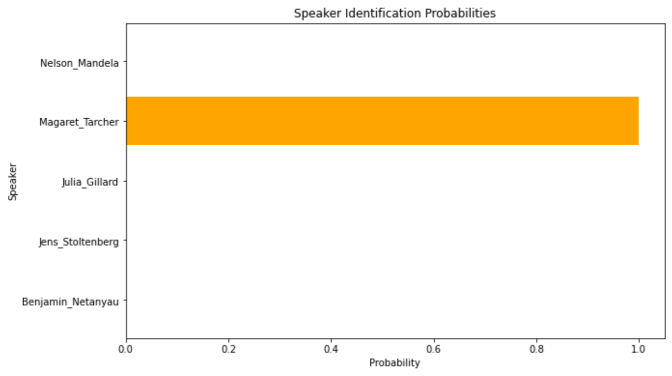


Fig. 9. Predicted Output

V. CONCLUSION AND FUTURE SCOPE

The present investigation delves deeply into the application of Convolutional Neural Networks (CNN) for speaker recognition, demonstrating the effectiveness of CNN in enhancing task accuracy. A notable improvement in speaker recognition accuracy has been attained by utilizing CNN. After 100 epochs, the model shows an accuracy rate of about 99.2%, indicating that it can handle a wide variety of input data and recognize individual speakers with accuracy. Moreover, CNN shows itself to be exceptionally skilled at obtaining features from audio spectrogram data, which permits a detailed and in-depth examination of the input. This feature makes it possible for the model to accurately capture each person's unique vocal traits, which significantly improves speaker recognition accuracy. Sophisticated methods like feature extraction and preprocessing are used in CNN implementations, and they greatly improve the quality and relevance of the input data used in the recognition process. This in turn helps explain the significant improvement in accuracy seen in the tasks involving speaker recognition. Results of several metrics, including model accuracy, model loss, precision recall curve, are provided. Additionally, it displays the expected result and its likelihood. Strategies like advanced architectures, hyperparameter tuning, ensemble learning, data augmentation, regularization techniques, transfer learning, normalization layers, attention mechanisms, and architecture search can be used to improve CNN architectures.

Acknowledgment

We would like to express our sincere gratitude to all our batchmates who have contributed to the successful completion of this project, Speaker Recognition using CNN. This endeavor would not have been possible without the unwavering support and guidance of Prof. Rutul Patel, whose expertise and encouragement played a pivotal role in shaping the project.

References

[1] R. Jagiasi, S. Ghosalkar, P. Kulal and A. Bharambe, ”CNN based speaker recognition in language and text-independent small scale system,” 2019 Third International conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC), Palladam, India, 2019, pp. 176-179, doi: 10.1109/I-SMAC47947.2019.9032667. keywords: Speaker recognition;Mel frequency cepstral coefficient;Feature extraction;Neural networks;Conferences;Training;Information technology;speaker recognition;neural network;voice sample;language-independent speaker recognition;independent speaker recognition system,

[2] Aroon, A., Dhonde, S. B. (2015). Speaker Recognition System using Gaussian Mixture Model. International Journal of Computer Applications, 130(14), 38–40. doi:10.5120/ijca2015907193

[3] Reynolds, D. A. (1995). Speaker identification and verification using Gaussian mixture speaker models. Speech Communication, 17(1-2), 91–108. doi:10.1016/0167-6393(95)00009-d

[4] Endres, W., Bambach, W., Flösser, G. (1971). Voice Spectrograms as a Function of Age, Voice Disguise, and Voice Imitation. The Journal of the Acoustical Society of America, 49(6B), 1842–1848. doi:10.1121/1.1912589

[5] Leggetter, C. J., Woodland, P. C. (1995). Maximum likelihood linear regression for speaker adaptation of continuous density hidden Markov models. Computer Speech Language, 9(2), 171–185.

[6] Campbell, W. M., Campbell, J. R., Reynolds, D. A., Jones, D. A., Leek, T. R. (n.d.). High-level speaker verification with support vector machines. 2004 IEEE International Conference on Acoustics, Speech, and Signal

[7] S. Ganvir and N. Lal, ”Automatic Speaker Recognition using Transfer Learning Approach of Deep Learning Models,” 2021 6th International Conference on Inventive Computation Technologies (ICICT), Coimbatore, India, 2021, pp. 595-601, doi: 10.1109/ICICT50816.2021.9358539. keywords: Deep learning;Transfer learning;Training data;Feature extraction;Data models;Speaker recognition;Human voice;Speaker recognition;Computer vision;Deep learning;Transfer learning;Performance comparison;One-shot learning

[8] J. Martinez, H. Perez, E. Escamilla and M. M. Suzuki, ”Speaker recognition using Mel frequency Cepstral Coefficients (MFCC) and Vector quantization (VQ) techniques,” CONIELECOMP 2012, 22nd International Conference on Electrical Communications and Computers, Cholula, Puebla, Mexico, 2012, pp. 248-251, doi: 10.1109/CONIELECOMP.2012.6189918. keywords: Mel frequency cepstral coefficient;Filter banks;Speaker recognition;Speech recognition;Databases;Speech;Vectors;Speech processing;Voice;speaker recognition;MFCC;Discrete Fourier Transform;Vector Quantization,

[9] A. Torfi, J. Dawson and N. M. Nasrabadi, ”Text-Independent Speaker Verification Using 3D Convolutional Neural Networks,” 2018 IEEE International Conference on Multimedia and Expo (ICME), San Diego, CA, USA, 2018, pp. 1-6, doi: 10.1109/ICME.2018.8486441. keywords: Feature extraction;Training;Computer architecture;Three-dimensional displays;Kernel;Convolutional neural networks;Adaptation models;Speaker verification;3D convolutional neural networks;text-independent;speaker model,

Appendix

Speaker Recognition using CNN

Model training

```
import os
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import average_precision_score
from tensorflow.keras.utils import plot_model
# Function to extract spectrogram features from audio files
def extract_spectrogram(file_path, n_mels=128, hop_length=512):
y, sr = librosa.load(file_path, sr=None)
mel_spec = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=n_mels, hop_length=hop_length)
mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
return mel_spec_db
# Function to preprocess audio data
def preprocess_data(data_path, n_mels=128, fixed_size=(128, 128)):
spectrograms = []
labels = []
```

```
speakers = os.listdir(data_path)
label_encoder = LabelEncoder()
for i, speaker in enumerate(speakers):
speaker_path = os.path.join(data_path, speaker)
for audio_file in os.listdir(speaker_path):
if audio_file.endswith(".wav"):
file_path = os.path.join(speaker_path, audio_file)
spectrogram = extract_spectrogram(file_path)
# Resize spectrogram to fixed size
new_spectrogram = np.zeros(fixed_size)
new_spectrogram[:spectrogram.shape[0], :spectrogram.shape[1]] = spectrogram[:fixed_size[0], :fixed_size[1]]
spectrograms.append(new_spectrogram)
labels.append(speaker)
spectrograms = np.array(spectrograms)
labels = np.array(labels)
# Convert labels to integers
labels_encoded = label_encoder.fit_transform(labels)
# Convert labels to one-hot encoding
labels_one_hot = to_categorical(labels_encoded)
return spectrograms, labels_one_hot, label_encoder
# Load and preprocess data
data_path = 'D:/Downloads/pcm_speeches'
spectrograms, labels, label_encoder = preprocess_data(data_path)
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(spectrograms, labels, test_size=0.2, random_state=42)
# Reshape input data to include batch dimension
X_train = np.expand_dims(X_train, axis=-1)
X_test = np.expand_dims(X_test, axis=-1)
# Define CNN model
```

```
model = Sequential([
Conv2D(64, kernel_size=(3, 3), activation='relu', input_shape=X_train.shape[1:]),
MaxPooling2D(pool_size=(2, 2)),
Conv2D(32, kernel_size=(3, 3), activation='relu'),
MaxPooling2D(pool_size=(2, 2)),
Conv2D(16, kernel_size=(3,3), activation='relu'),
MaxPooling2D(pool_size=(2, 2)),
Flatten(),
Dense(128, activation='relu'),
Dense(64, activation='relu'),
Dropout(0.5),
Dense(len(label_encoder.classes_), activation='softmax')
])
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Train the model
epochs = 5
history = model.fit(X_train, y_train, epochs=epochs, batch_size=32, validation_split=0.2, verbose=1)
# Evaluate the model
loss, accuracy = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
model.save("D:/Nirma/Sem 6/ML/spass/speaker_recognition_model.h5")
# Plot training history
plt.figure(figsize=(12, 6))
# Plot accuracy
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Model Accuracy')
plt.legend()
# Plot loss
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Model Loss')
plt.legend()
plt.tight_layout()
plt.show()
# Calculate and plot precision and recall curves
y_pred = model.predict(X_test)
average_precision = average_precision_score(y_test, y_pred)
plt.figure()
precision = dict()
recall = dict()
for i in range(len(label_encoder.classes_)):
precision[i], recall[i], _ = precision_recall_curve(y_test[:, i], y_pred[:, i])
plt.plot(recall[i], precision[i], lw=2, label='Class {}'.format(label_encoder.classes_[i]))
plt.xlabel("Recall")
plt.ylabel("Precision")
plt.title("Precision-Recall curve")
plt.legend(loc="best")
plt.show()
plot_model(model, to_file='model_architecture.png', show_shapes=True,
show_layer_names=True)
```

Speaker Testing

```
import os
import numpy as np
import matplotlib.pyplot as plt
import librosa
import librosa.display
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.models import load_model
# Function to extract spectrogram features from audio files
def extract_spectrogram(file_path, n_mels=128, hop_length=512):
y, sr = librosa.load(file_path, sr=None)
mel_spec = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=n_mels, hop_length=hop_length)
mel_spec_db = librosa.power_to_db(mel_spec, ref=np.max)
return mel_spec_db
# Function to preprocess audio data
def preprocess_data(spectrogram, fixed_size=(128, 128)):
# Resize spectrogram to fixed size
label_encoder = LabelEncoder()
new_spectrogram = np.zeros(fixed_size)
new_spectrogram[:spectrogram.shape[0], :spectrogram.shape[1]] = spectrogram[:fixed_size[0], :fixed_size[1]]
spectrogram = np.expand_dims(new_spectrogram, axis=-1)
return spectrogram
# Load trained model
model_path = 'D:/Nirma/Sem 6/ML/spass/speaker_recognition_model.h5' # Update with your model path
model = load_model(model_path)
# Function to predict speaker from audio file
def predict_speaker(audio_file, model, label_encoder):
# Extract spectrogram features
```

```
spectrogram = extract_spectrogram(audio_file)
# Preprocess spectrogram
spectrogram = preprocess_data(spectrogram)
# Predict speaker
probabilities = model.predict(np.expand_dims(spectrogram, axis=0))[0]
predicted_label_index = np.argmax(probabilities)
predicted_speaker = label_encoder.classes_[predicted_label_index]
return predicted_speaker, probabilities
# Function to plot probabilities
def plot_probabilities(probabilities, label_encoder):
num_speakers = len(label_encoder.classes_)
plt.figure(figsize=(10, 6))
plt.barh(range(num_speakers), probabilities, color='orange')
plt.yticks(range(num_speakers), label_encoder.classes_)
plt.xlabel('Probability')
plt.ylabel('Speaker')
plt.title('Speaker Identification Probabilities')
plt.show()
# Test with an audio file
audio_file_path = 'D:/Downloads/pcm_speeches/Magaret_Tarcher/4.wav' #
Update with your audio file path
predicted_speaker, probabilities = predict_speaker(audio_file_path, model,
label_encoder)
print(f"Predicted Speaker: {predicted_speaker}")
plot_probabilities(probabilities, label_encoder)
```