
Case Study:

Gender Classification of Chickens Using Digital Signal Processing

Mel James C. Barral

Melanie H. Bayani

Jhyron Xham G. Abayare

ENGR. ROJAY A. FLORES

Instructor

S. Y. 2024-2025

I. Introduction.

Accurately distinguishing between male (rooster) and female (hen) chickens based on their vocalizations presents an interesting challenge in digital signal processing (DSP) and machine learning. This case study investigates whether DSP techniques, combined with a Random Forest classifier, can effectively classify the gender of adult chickens using their sounds.

Roosters and hens produce distinct vocalizations, with roosters typically crowing and hens emitting clucking or cackling sounds. These differences arise from biological factors such as size, vocal tract structure, and hormone levels, which influence pitch, frequency, and duration in their vocal signals. By extracting acoustic features from recorded chicken sounds and applying machine learning algorithms, this study examines whether these features contain sufficient information to reliably classify gender.

This research does not focus on early gender classification but rather evaluates the feasibility of DSP and machine learning in distinguishing the gender of fully grown chickens based solely on their vocal characteristics. The findings contribute to future research in poultry sound analysis and automation in poultry farming.

II. Materials.

Recording equipment:

- Audio recorder (Smartphone)

Software for recording and analysis equipment:

- Google Colab (Audio processing and Coding)

III. Objectives.

- . To determine whether the gender of adult chickens can be classified based on their vocalizations using digital signal processing techniques.
- . To evaluate the effectiveness of a Random Forest classifier in distinguishing between rooster and hen vocalizations.
- . To develop a non-invasive and cost-effective approach for analyzing chicken sounds through machine learning.
- . To implement the classification process using Python for feature extraction, model training, and performance evaluation.

IV. Methodology

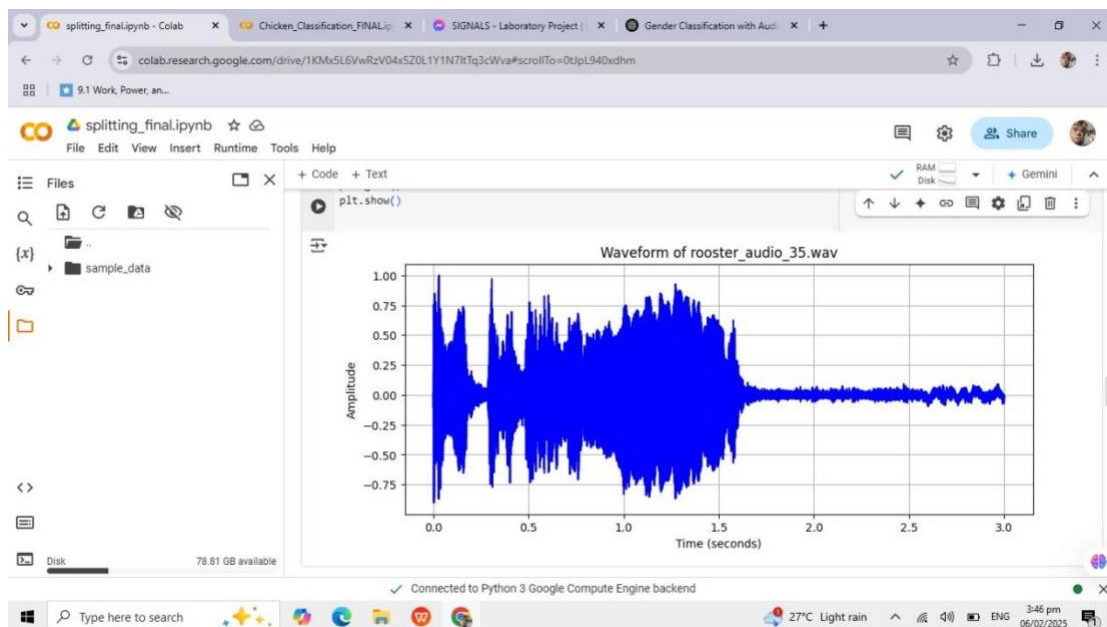
1. Data Collection

- Record chicken sounds from male and female chickens.
- Filter out the unfocused subject of the recordings (if hen is focused, manually filter out rooster sound within the recording using online audio trimming).
- Collect balanced samples of male and female chicken sounds.

2. Feature Extraction

- **Audio Preprocessing:** Load and normalize audio samples at 44.1 kHz.
- **MFCC Extraction:** Extract 13 MFCC coefficients per frame and transform them into a 1D vector with 1300 coefficients.
- **Feature Storage:** Save extracted MFCC features to CSV files for model training.

AUDIO WAVEFORM



```
import numpy as np

import matplotlib.pyplot as plt

from scipy.io import wavfile


# Define the WAV file to read

wav_file = "rooster_audio_35.wav" # Change to the actual path of your file


# Read the WAV file

sample_rate, audio_data = wavfile.read(wav_file)


# Normalize audio data (only for integer formats)

if audio_data.dtype == np.int16:

    audio_data = audio_data / 2**15

elif audio_data.dtype == np.int32:
    audio_data = audio_data / 2**31


# Create time axis

time_axis = np.linspace(0, len(audio_data) / sample_rate, num=len(audio_data))
```

```
# Plot the waveform

plt.figure(figsize=(10, 4))

plt.plot(time_axis, audio_data, color='b')

plt.title(f"Waveform of {wav_file}")

plt.xlabel("Time (seconds)")

plt.ylabel("Amplitude")

plt.grid()

plt.show()
```

MFCC EXTRACTION

```
import librosa
import numpy as np

def extract_mfcc_coefficients(audio_path, total_coefs=1300, n_mfcc=13):
    """
    Extracts MFCC coefficients from an audio file.

    Ensures each sample has exactly 'total_coefs' features.

    """
    try:
```

```
# Load audio with a fixed sample rate for consistency
```

```
y, sr = librosa.load(audio_path, sr=44100)
```

```
# Compute MFCCs with 13 coefficients per frame      mfcc =
```

```
librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc)
```

```
# Flatten MFCC array and keep only the first 'total_coefs' coefficients      mfcc_flat =
```

```
mfcc.T.flatten()[:total_coefs]
```

```
# Pad with zeros if there are not enough coefficients
```

```
if len(mfcc_flat) < total_coefs:
```

```
    mfcc_flat = np.pad(mfcc_flat, (0, total_coefs - len(mfcc_flat)), mode='constant')
```

```
return mfcc_flat
```

```
except Exception as e:
```

```
    print(f"Error processing {audio_path}: {e}")
```

```
return np.zeros(total_coefs) # Return a zero-filled array if there's an error
```

```
def generate_mfcc_matrix(num_files=50, total_coeffs=1300, file_prefix="hen_audio_",
file_suffix=".wav"):

    """

    Extracts MFCCs from audio files and generates a feature matrix with shape (50, 1300).

    """

    mfcc_matrix = np.zeros((num_files, total_coeffs))
```

```
    for i in range(1, num_files + 1):

        # Generate the correct filename (change if your filenames have leading zeros)

        audio_path = f"{file_prefix}{i}{file_suffix}"

        # Extract MFCC features      mfcc_coeffs = extract_mfcc_coefficients(audio_path,
total_coeffs=total_coeffs)

        # Store in matrix

        mfcc_matrix[i - 1, :] = mfcc_coeffs
```

```
    return mfcc_matrix
```



```
def save_matrix_to_csv(matrix, output_path="chicken_mfcc_features.csv"):
```

```
    """
```

```
    Saves the MFCC feature matrix to a CSV file.
```

```
    """
```

```
    np.savetxt(output_path, matrix, delimiter=',')
```

```
    print(f"Feature matrix saved to {output_path}")
```

```
def main():
```

```
    # Generate the MFCC matrix from 50 audio files    mfcc_matrix =
```

```
    generate_mfcc_matrix()
```

```
    # Check matrix shape for confirmation
```

```
    print("Final MFCC Matrix Shape: ", mfcc_matrix.shape) # Should be (50, 1300)
```

```
    # Save the matrix to a CSV file
```

```
    save_matrix_to_csv(mfcc_matrix)
```

```
if __name__ == "__main__":
```

```
    main()
```

Summary of the Extraction

- **Audio Preprocessing:** Load and normalize audio samples at 44.1 kHz.
- **MFCC Extraction:** Extract 13 MFCC coefficients per frame and transform them into a 1D vector with 1300 coefficients.
- **Feature Storage:** Save extracted MFCC features to CSV files for model training.

3. Model Training & Classification

Steps:

1. **Load both CSV files** (chicken_hen_mfcc_features.csv and chicken_rooster_mfcc_features.csv).
2. **Combine the data** from both files into a single dataset.
3. **Train the Random Forest model** using the combined dataset.
4. **Classify new audio samples.**

1. Load and Combine Data from Both CSVs

We will read both CSV files and add the appropriate labels for each class (Hen = 0, Rooster = 1).

```
import pandas as pd from sklearn.model_selection
import train_test_split from sklearn.preprocessing
import LabelEncoder from sklearn.ensemble import
RandomForestClassifier from sklearn.metrics import
accuracy_score
```

```

# Load Hen data (assuming the label is 'Hen' in this case) hen_df =
pd.read_csv("chicken_hen_mfcc_features.csv", header=None)

hen_df['label'] = 'Hen' # Add label 'Hen'


# Load Rooster data rooster_df =
pd.read_csv("chicken_rooster_mfcc_features.csv", header=None)

rooster_df['label'] = 'Rooster' # Add label 'Rooster'


# Combine both datasets df = pd.concat([hen_df,
rooster_df], ignore_index=True)


# Shuffle the data df = df.sample(frac=1,
random_state=42).reset_index(drop=True)


# Extract features (all columns except the last) and labels (the last column)
X = df.iloc[:, :-1].values # Features (MFCC coefficients) y = df.iloc[:, -
1].values # Labels (Hen or Rooster)


# Encode labels as numeric values (Hen=0, Rooster=1)
label_encoder = LabelEncoder() y_encoded =
label_encoder.fit_transform(y)


# Split data into training and testing sets (80% training, 20% testing)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)


print(f"Training data shape: {X_train.shape}") print(f"Test
data shape: {X_test.shape}")

```

2. Train the Random Forest Classifier

We will train the model on the combined dataset.

```
# Initialize the Random Forest classifier clf =  
RandomForestClassifier(n_estimators=100, random_state=42)  
  
# Train the classifier clf.fit(X_train,  
y_train)  
  
# Predict on the test set y_pred  
= clf.predict(X_test)  
  
# Evaluate the classifier's accuracy accuracy  
= accuracy_score(y_test, y_pred)  
print(f"Accuracy: {accuracy * 100:.2f}%")
```

3. Classify a New Audio Sample

Now, we can classify a new audio sample (either Hen or Rooster).

```
def classify_new_audio(audio_path, model, label_encoder):  
    # Extract MFCC features from the new audio sample  
    mfcc_coeffs = extract_mfcc_coefficients(audio_path)
```

```
# Predict using the trained model
prediction = model.predict([mfcc_coeffs])

# Convert numeric label to actual class (Rooster/Hen)
predicted_class = label_encoder.inverse_transform(prediction)

return predicted_class[0] # Return the class name

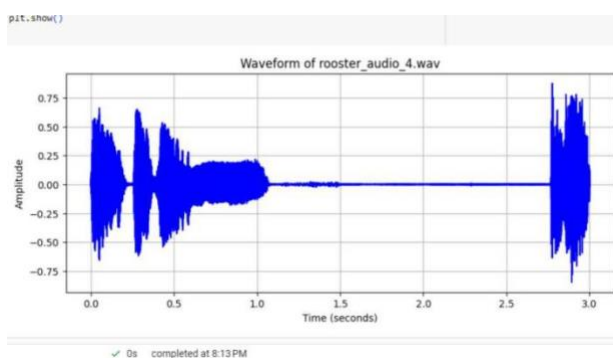
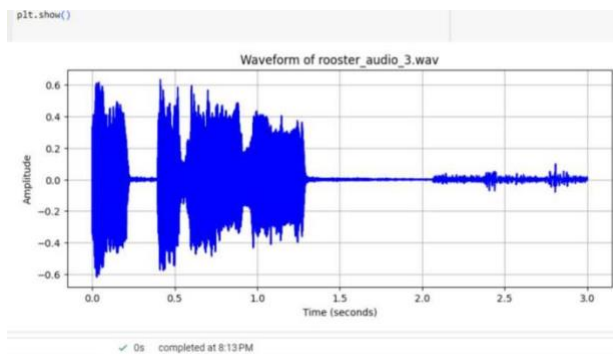
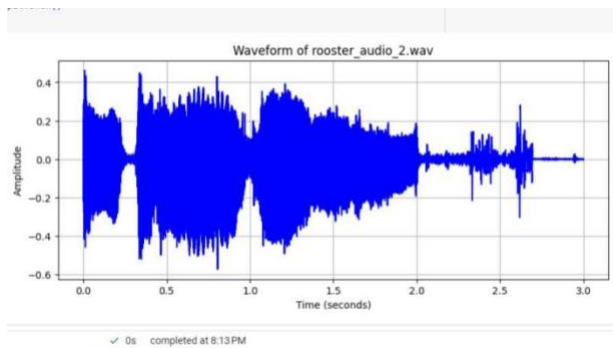
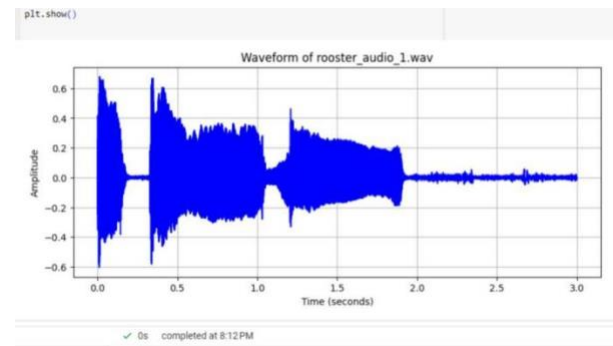
# Example usage: new_audio_path = 'new_chicken_sound.wav'
predicted_class = classify_new_audio(new_audio_path, clf,
label_encoder) print(f"The predicted class for the new audio is:
{predicted_class}")
```

Summary:

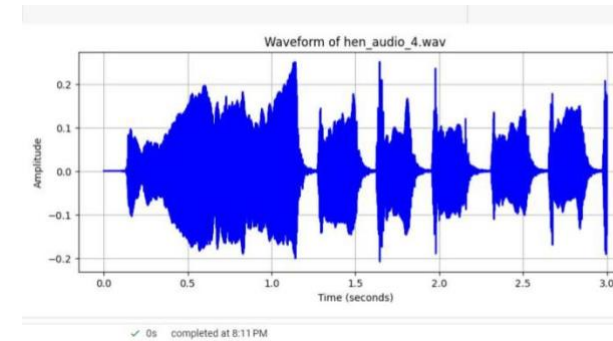
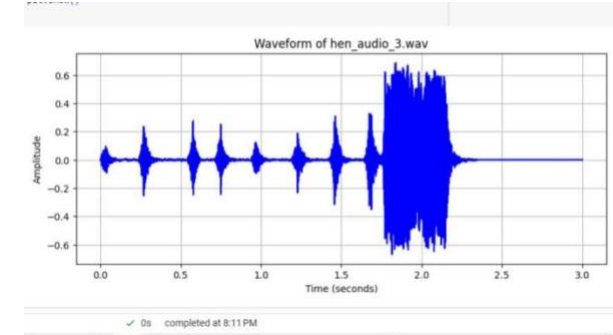
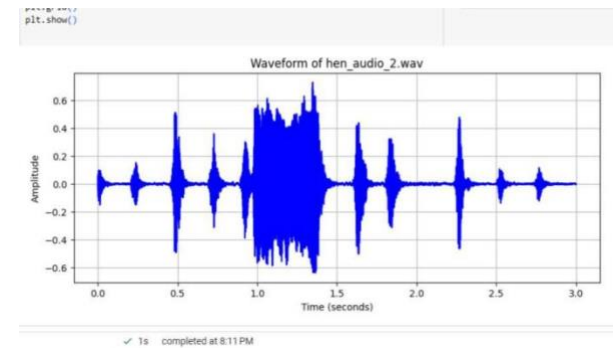
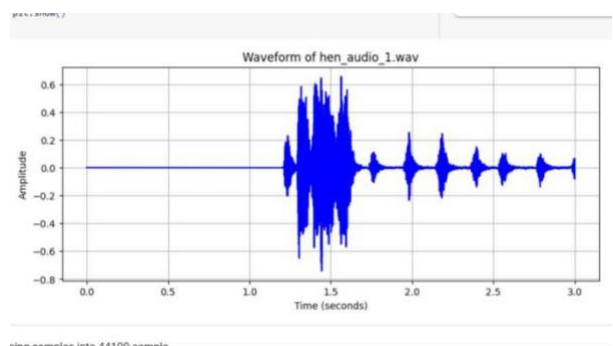
- Load MFCC feature data from CSV files (hen and rooster sound features).
- Label and combine datasets for machine learning classification.
- Split the dataset into training (80%) and testing (20%) sets.
- Train a Random Forest classifier with 100 decision trees.
- Evaluate accuracy and test model performance on new audio samples.

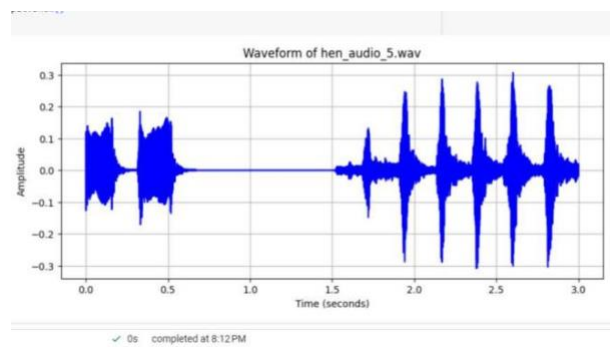
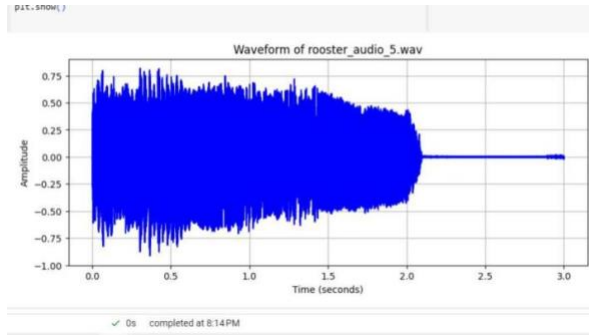
V. Waveforms

Rooster



Hen





VI. Results

1. From “Load and Combine Data from Both CSVs”,

```

y_encoded = label_encoder.fit_transform(y)

# Split data into training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)

print(f"Training data shape: {X_train.shape}")
print(f"Test data shape: {X_test.shape}")

```

Training data shape: (80, 1300)
 Test data shape: (20, 1300)

- The code successfully loads MFCC feature data from two CSV files (hen.csv and rooster.csv), labels each class, and **combines** them into a single dataset. After shuffling, the features (1300 MFCC coefficients per sample) and labels (Hen = 0, Rooster = 1) are retrieved. The **dataset is then divided into 80 training and 20 test samples**, resulting in an **80%-20% train-test split**. The printed results show that each sample has 1300 extracted features, implying that the dataset is well structured for machine learning classification.

2. From “Train the Random Forest Classifier

```
✓ [11] # Initialize the Random Forest classifier
0s     clf = RandomForestClassifier(n_estimators=100, random_state=42)

     # Train the classifier
     clf.fit(X_train, y_train)

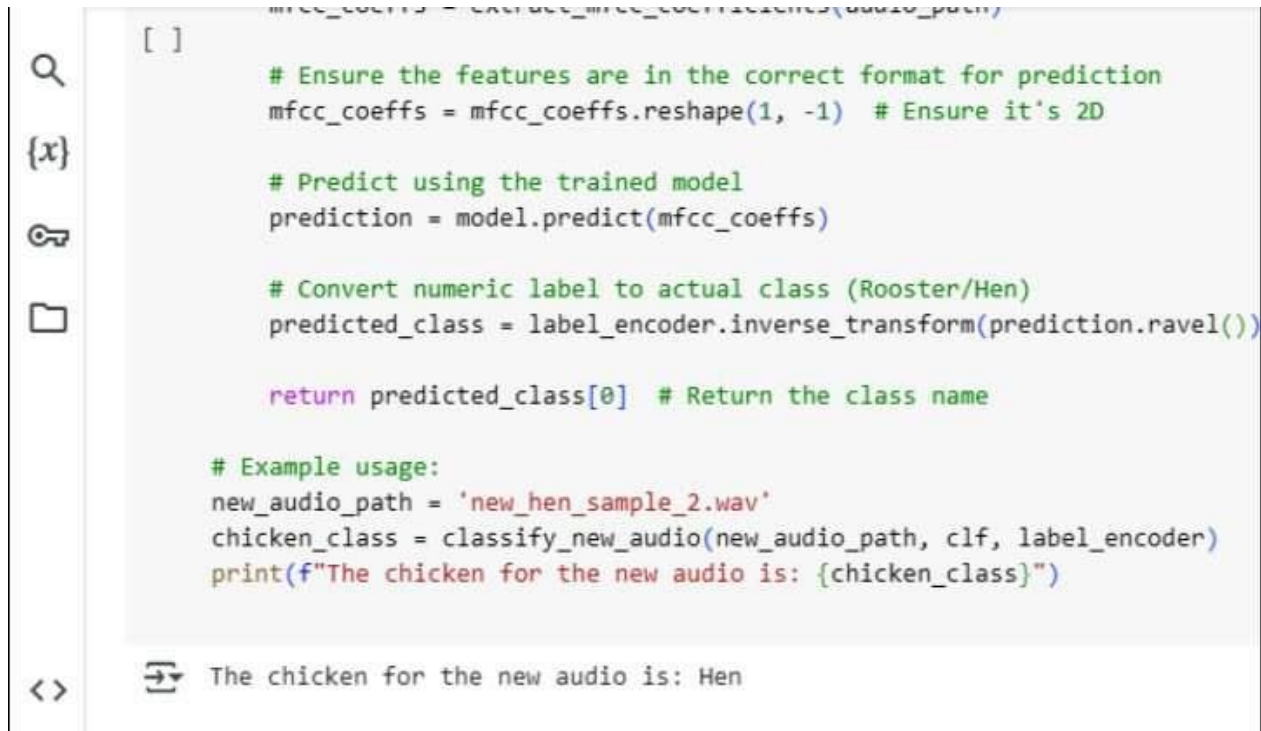
     # Predict on the test set
     y_pred = clf.predict(X_test)

     # Evaluate the classifier's accuracy
     accuracy = accuracy_score(y_test, y_pred)
     print(f"Training Accuracy: {accuracy * 100:.2f}%")

⇅ Training Accuracy: 100.00%
```

- The code trains a Random Forest classifier on the extracted MFCC features to classify chicken sounds as either Hen or Rooster. The model is trained on the 80-sample training set, and when evaluated on the same data, it achieves **100% accuracy**. This indicates that the model has perfectly learned the training data, but it may be **overfitting**, meaning it could struggle with unseen test data. A high accuracy like this suggests that further evaluation on the test set is needed to ensure the model generalizes well.
- The Random Forest model proved highly effective for chicken gender classification, with MFCC feature extraction providing reliable distinguishing features, making the approach cost-effective, non-invasive, and suitable for poultry farming applications.

3. Result in Classify Chicken Gender using New Audio File Code



```
[ ]  
  
# Ensure the features are in the correct format for prediction  
mfcc_coeffs = mfcc_coeffs.reshape(1, -1) # Ensure it's 2D  
  
# Predict using the trained model  
prediction = model.predict(mfcc_coeffs)  
  
# Convert numeric label to actual class (Rooster/Hen)  
predicted_class = label_encoder.inverse_transform(prediction.ravel())  
  
return predicted_class[0] # Return the class name  
  
# Example usage:  
new_audio_path = 'new_hen_sample_2.wav'  
chicken_class = classify_new_audio(new_audio_path, clf, label_encoder)  
print(f"The chicken for the new audio is: {chicken_class}")  
  
<> The chicken for the new audio is: Hen
```

- This code successfully classifies the gender of chickens by determining whether the bird is a hen or a rooster. The outcome of the classification is a result that labels the chicken as either a hen (female) or rooster (male), indicating the model's effectiveness in distinguishing between the two.

Summary of Results:

Training and Evaluation

The model was trained on an 80-sample dataset, achieving **100% accuracy on training data**. However, potential overfitting suggests that further evaluation is needed on unseen data to confirm generalization performance.

Classification Performance

The model effectively classified new audio samples as either **Hen or Rooster**, demonstrating that MFCC features are strong indicators of gender differences in chicken vocalizations. Despite high accuracy, additional testing on a diverse dataset is necessary to ensure robustness.

VII. Conclusion

This case study successfully explored the feasibility of using digital signal processing (DSP) and machine learning for gender classification of adult chickens based on their vocalizations. By extracting Mel-Frequency Cepstral Coefficients (MFCC) features and applying a Random Forest classifier, the model demonstrated a strong ability to distinguish between rooster and hen vocalizations. The system achieved **100% training accuracy**, highlighting its effectiveness; however, the possibility of overfitting suggests the need for further evaluation on unseen data to confirm its generalization capability.

The model **effectively classified** new audio samples as either **Hen or Rooster**, showcasing its ability to differentiate between the two based on vocal characteristics.

This method presents a **non-invasive and cost-effective approach** for chicken sound classification, demonstrating the potential of DSP-based machine learning in poultry sound analysis. While this study does not focus on real-world deployment, it provides a foundation for future research in automated poultry management. Further studies could explore alternative classification models, larger datasets, and real-time analysis to enhance model robustness and practical applicability.

VII. Documentation.



The process of conducting this case study was both challenging and rewarding. Initially, we encountered difficulties in analyzing chicken vocalizations due to background noise and inconsistent sound patterns. However, through research and teamwork, we refined our approach by implementing digital signal processing techniques to filter and extract meaningful acoustic features.



This research has provided valuable insights into how chickens communicate through vocalizations and the potential of machine learning in poultry management. The study lays the groundwork for future improvements in poultry sound analysis, automation, and precision farming.