Configuration Manual

MSc Research Project
MSc Data Analytics

Ashraf Hussain Raheem Basha
Student ID: x23191899

School of Computing
National College of Ireland

Supervisor: Vikas Tomer

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Ashraf Hussain Raheem Basha
Student ID:	X23191899
Programme:	MSc. Data Analytics — **Year:** 2024
Module:	Research Project Configuration Manual
Lecturer:	Vikas Tomer
Submission Due Date:	12 December 2024
Project Title:	Enhancing Crime Prevention via Human Scream Detection with Deep Learning and Machine Learning
Word Count:	600 — **Page Count:** 11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ashraf Hussain Raheem Basha

Date: 12 December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ashraf Hussain Raheem Basha
x23191899

1 Introduction

The objective of this document specifies the machine configuration required for replicating the scream detection models. This configuration manual provides step-by-step instructions for setting up, running, and troubleshooting the scream detection project.

2 Device Specification

2.1 Hardware Requirements

The below image represents the specifications of the device on which the project was carried out.

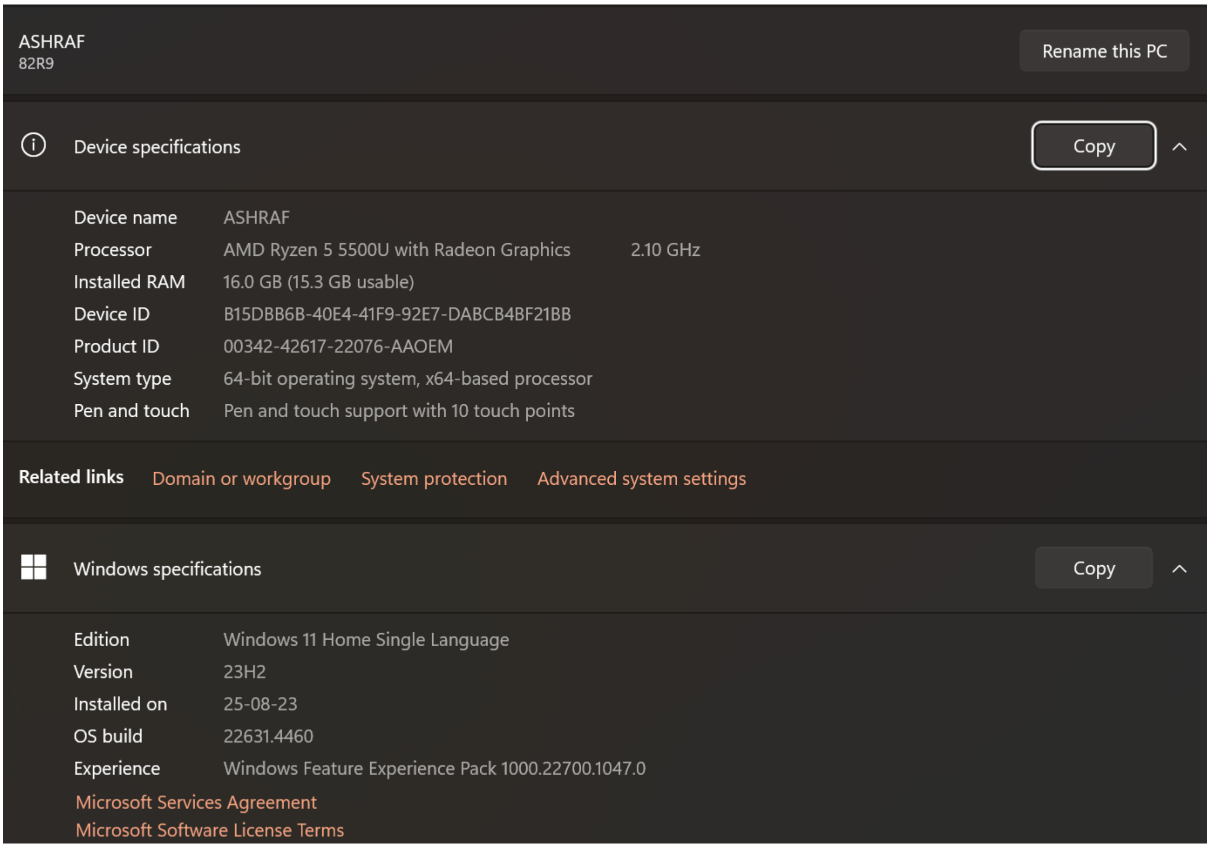

Figure 1 - Device Specification

2.2 Software Specification

- **Operating System**: Windows 10/11, macOS, or Linux
- **Python Version**: Python 3.10 or above

- Download & Install Anaconda Navigator as it is highly recommended for carrying out the project. It contains Jupyter Notebook and necessary python libraries and configurations required for running the script smoothly.

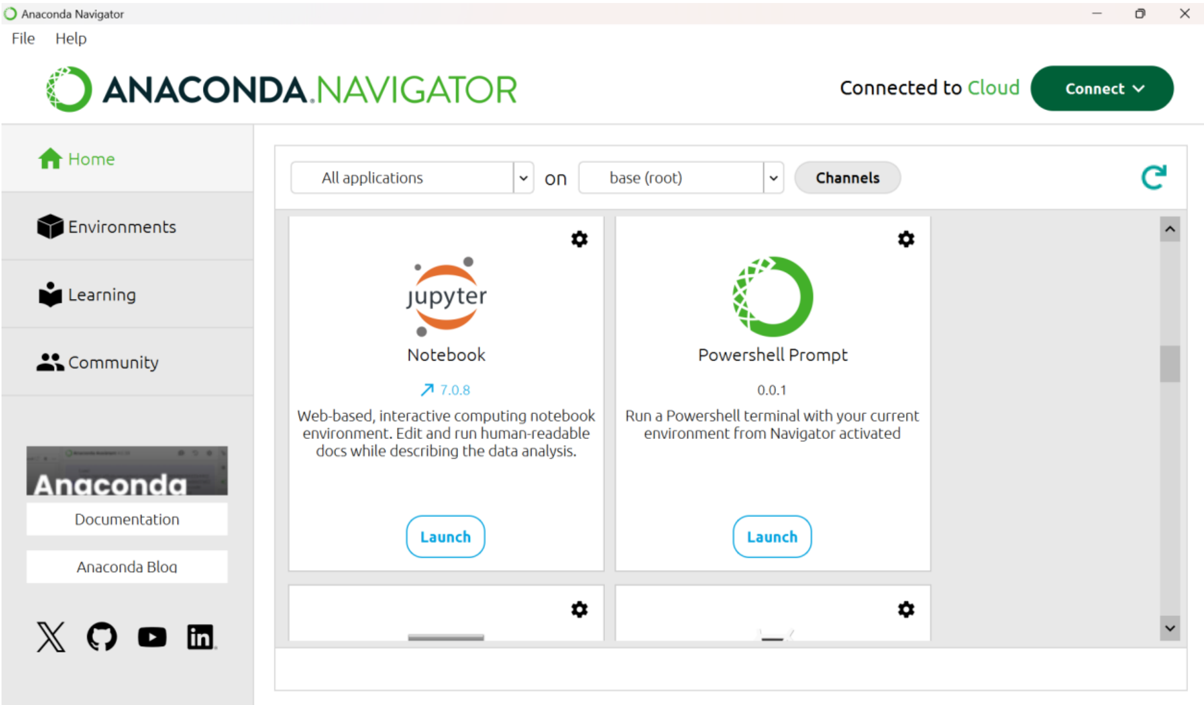

Figure 2 – Anaconda Navigator

3 Dataset

The dataset for this project was collected from publicly available sources such as AudioSet and Freesound. The link for the sources is provided below:

Source 1 – https://research.google.com/audioset/download.html
Source 2 - https://freesound.org/

The audio file containing human scream and non-scream human scream such as environment, vehicle, non-human sounds.

4 Implementation of models

4.1 Python Libraries

The Anaconda platform contains all the python libraries. Install the required python libraries and import them in the project.

```python
import os
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.utils import resample
import numpy as np
import librosa
from collections import Counter
import random
import librosa.display
import soundfile as sf
from tqdm import tqdm
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report, confusion_matrix
import joblib
import pandas as pd
from sklearn.neural_network import MLPClassifier
from torchvision import datasets, transforms, models
import torch
import torch.nn as nn
from torch.utils.data import DataLoader
from sklearn.metrics import ConfusionMatrixDisplay
```

Figure 3 – Libraries used

4.2 Load Dataset

The dataset must be loaded for further process in the project. The audio files are stored in a directory and separated as scream and non-scream and labelled as ‘yes’ and ‘no’ respectively.

```python
scream_dir = r"D:\Ashraf\NCI\Semester 2\Research In Computing\Dataset\Screaming"
non_scream_dir = r"D:\Ashraf\NCI\Semester 2\Research In Computing\Dataset\NotScreaming"

file_paths = []
labels = []

for file_name in os.listdir(scream_dir):
    file_paths.append(os.path.join(scream_dir, file_name))
    labels.append("yes")

for file_name in os.listdir(non_scream_dir):
    file_paths.append(os.path.join(non_scream_dir, file_name))
    labels.append("no")

print(f"Loaded {len(file_paths)} audio files.")
print(f"Number of files: {len(file_paths)}")
print(f"Labels count: {len(labels)}")
print(f"Scream files labeled 'yes': {labels.count('yes')}")
print(f"Non-scream files labeled 'no': {labels.count('no')}")
```

Figure 4 – Dataset Loading

4.3 Load Dataset

Since the dataset is imbalanced, they are balanced by up sampling the low class matching the number of higher class. So that both the classes can be treated equally by the model.

```python
yes_indices = [i for i, label in enumerate(labels) if label == 'yes']
no_indices = [i for i, label in enumerate(labels) if label == 'no']

yes_upsampled = resample(
    yes_indices,
    replace=True,
    n_samples=len(no_indices),
    random_state=42
)

balanced_indices = yes_upsampled + no_indices
np.random.shuffle(balanced_indices)
```

Figure 5 – Sampling

4.4 Data Augmentation

Transform the dataset as shown in Figure 6 by changing the time, frequency, speed and adding noise to the original file. This helps the model to learn about the screams and background noises available in the audio file and recognise them as non-scream audio.

```python
# Directory to store the transformed data
augmented_dir = r"D:\Ashraf\NCI\Semester 3\Code\augmented_data_new"
os.makedirs(augmented_dir, exist_ok=True)

# Functions for transformation
def time_shift(audio, shift_max=0.2):
    shift = int(np.random.uniform(-shift_max, shift_max) * len(audio))
    return np.roll(audio, shift)

def pitch_shift(audio, sample_rate, n_steps):
    return librosa.effects.pitch_shift(audio, sr=sample_rate, n_steps=n_steps)

def speed_change(audio, speed_factor):
    return librosa.effects.time_stretch(audio, rate=speed_factor)

def add_noise(audio, noise_factor=0.005):
    noise = np.random.randn(len(audio))
    noisy_audio = audio + noise_factor * noise
    return noisy_audio / np.max(np.abs(noisy_audio))

def augment_audio(audio, sample_rate, file_name, label, augmentations=None, save_dir=augmented_dir):
    os.makedirs(save_dir, exist_ok=True)
    if augmentations is None:
        augmentations = ['time_shift', 'pitch_shift', 'speed_change', 'add_noise']

    # Variables to store the augmented values
    augmented_file_paths = []
    augmented_labels = []

    for augmentation in augmentations:
        save_path = os.path.join(save_dir, f"{file_name}_{augmentation}.wav")
```

```python
        # Skip if the file already exist
        if os.path.exists(save_path):
            augmented_file_paths.append(save_path)
            augmented_labels.append(label)
            continue

        if augmentation == "time_shift":
            augmented = time_shift(audio)
        elif augmentation == "pitch_shift":
            n_steps = np.random.randint(-2, 3)
            augmented = pitch_shift(audio, sample_rate, n_steps)
        elif augmentation == "speed_change":
            speed_factor = np.random.uniform(0.9, 1.1)
            augmented = speed_change(audio, speed_factor)
        elif augmentation == "add_noise":
            augmented = add_noise(audio)

        sf.write(save_path, augmented, sample_rate)
        augmented_file_paths.append(save_path)
        augmented_labels.append(label)

    return augmented_file_paths, augmented_labels

file_paths_augmented = []
labels_augmented = []

for file_path, label in tqdm(zip(file_paths_balanced, labels_balanced), total=len(file_paths_balanced)):
    audio, sample_rate = librosa.load(file_path, sr=44100)
    file_name = os.path.splitext(os.path.basename(file_path))[0]
    augmented_file_paths, augmented_labels = augment_audio(audio, sample_rate, file_name, label, save_dir=augmented_dir)
    file_paths_augmented.extend(augmented_file_paths)
```

Figure 6 – Data Augmentation

4.5 Feature Extraction

Extract the audio data into numerical features for better analysis as shown in Figure 7. Extraction uses MFCCs, Chroma Features, Spectral Contrast and Zer-Crossing Rate for the machine learning model to learn better.

For deep learning models, the audio files are converted to spectrogram images since ResNet model can learn better with ImageNet. This can be done as show in Figure 8.

```
features_file = "features.npy"
labels_file = "feature_labels.npy"

def extract_features(file_paths, labels, sr=44100, n_mfcc=40):
    features = []
    feature_labels = []

    for file_path, label in tqdm(zip(file_paths, labels), total=len(file_paths), desc="Extracting Features"):
        try:
            audio, sample_rate = librosa.load(file_path, sr=sr)

            # Features to be extracted
            mfccs = np.mean(librosa.feature.mfcc(y=audio, sr=sample_rate, n_mfcc=n_mfcc).T, axis=0)
            chroma = np.mean(librosa.feature.chroma_stft(y=audio, sr=sample_rate).T, axis=0)
            spectral_contrast = np.mean(librosa.feature.spectral_contrast(y=audio, sr=sample_rate).T, axis=0)
            zero_crossing = np.mean(librosa.feature.zero_crossing_rate(y=audio))

            # Vectorize the features
            feature_vector = np.concatenate([mfccs, chroma, spectral_contrast, [zero_crossing]])
            features.append(feature_vector)
            feature_labels.append(label)

        # Log the corrupted file
        except Exception as e:
            print(f"Error processing {file_path}: {e}")
            continue

    return np.array(features), np.array(feature_labels)

if not os.path.exists(features_file) or not os.path.exists(labels_file):
    features, feature_labels = extract_features(file_paths_augmented, labels_augmented)

    # Save the features and labels for future use
    np.save(features_file, features)
    np.save(labels_file, feature_labels)
    print(f"Features saved to {features_file} and labels saved to {labels_file}")
else:
    features = np.load(features_file)
    feature_labels = np.load(labels_file)
    print(f"Loaded features from {features_file} and labels from {labels_file}")

print(f"Number of features: {features.shape[0]}")
print(f"Unique labels: {np.unique(feature_labels)}")
```

Figure 7 – Feature Extraction

```
# Directory to save spectogram imgaes
image_dir = r"D:\Ashraf\NCI\Semester 3\Code\balanced_spectrogram_images"
os.makedirs(image_dir, exist_ok=True)

os.makedirs(os.path.join(image_dir, "yes"), exist_ok=True)
os.makedirs(os.path.join(image_dir, "no"), exist_ok=True)

# Function to create and save the spectogram images
def save_mel_spectrogram_images(file_paths, labels, image_dir):
    processed_yes = 0
    processed_no = 0
    error_files = []

    for file_path, label in tqdm(zip(file_paths, labels), total=len(file_paths), desc="Generating Spectrograms"):
        try:
            label_dir = os.path.join(image_dir, label)
            os.makedirs(label_dir, exist_ok=True)
            save_path = os.path.join(label_dir, os.path.basename(file_path).replace('.wav', '.png'))

            if os.path.exists(save_path) and os.path.getsize(save_path) > 0:
                continue

            audio, sample_rate = librosa.load(file_path, sr=44100)

            mel_spectrogram = librosa.feature.melspectrogram(y=audio, sr=sample_rate, n_mels=128, fmax=8000)
            mel_db = librosa.power_to_db(mel_spectrogram, ref=np.max)
```

```
            plt.figure(figsize=(4, 4))
            librosa.display.specshow(mel_db, sr=sample_rate, x_axis='time', y_axis='mel', cmap='viridis')
            plt.axis('off')
            plt.savefig(save_path, bbox_inches='tight', pad_inches=0)
            plt.close()

            if label == "yes":
                processed_yes += 1
            elif label == "no":
                processed_no += 1

        # Log the error files
        except Exception as e:
            print(f"Error processing file {file_path}: {e}")
            error_files.append(file_path)

    return processed_yes, processed_no, error_files
# Process the spectogram creation in batches since the data is huge
batch_size = 1000
total_processed_yes = 0
total_processed_no = 0
all_error_files = []

for i in range(0, len(file_paths_augmented), batch_size):
    batch_files = file_paths_augmented[i:i+batch_size]
    batch_labels = labels_augmented[i:i+batch_size]
    processed_yes, processed_no, error_files = save_mel_spectrogram_images(batch_files, batch_labels, image_dir)

    total_processed_yes += processed_yes
    total_processed_no += processed_no
    all_error_files.extend(error_files)

    print(f"Batch {i // batch_size + 1}/{(len(file_paths_augmented) + batch_size - 1) // batch_size} completed.")
    print(f"YES spectrograms in this batch: {processed_yes}")
    print(f"NO spectrograms in this batch: {processed_no}")
    print(f"Errors in this batch: {len(error_files)}")

print(f"Total YES spectrograms generated: {total_processed_yes}")
print(f"Total NO spectrograms generated: {total_processed_no}")
print(f"Failed to process {len(all_error_files)} files.")

# Log the error files
if all_error_files:
    print("Problematic files:")
    for error_file in all_error_files[:10]:
        print(error_file)
```

Figure 8 – Spectrogram Generation

4.6 Data Splitting

Split the data into training, validation and testing. This split allows having reliable evaluation metrics for each subset as well as avoiding data leakage between them.

```
X_train_val, X_test, y_train_val, y_test = train_test_split(features, feature_labels, test_size=0.15, stratify=feature_labe
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.1765, stratify=y_train_val, random_s
```

Figure 9 – Data Splitting

Values for data Splitting:

features, feature_labels, test_size=0.15, stratify=feature_labels, random_state=42

X_train_val, y_train_val, test_size=0.1765, stratify=y_train_val, random_state=42

4.7 Standardization

Scale the feature data to ensure consistent numerical ranges, which is critical for many machine learning models to perform optimally.

```
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)
```

Figure 10 - Standardization

4.8 Model Implementation and Evaluation

4.8.1 Support Vector Machine

Train the SVM model with linear kernel.

```
print("Training SVM...")
svm_model = SVC(probability=True, kernel="linear", random_state=42)
svm_model.fit(X_train_scaled, y_train)
```

Figure 11 – SVM Training

The trained model is then validated with the evaluation metrics such as accuracy, precision, recall, f1-score, specificity and balanced accuracy.

```
y_train_pred_svm = svm_model.predict(X_train_scaled)
y_val_pred_svm = svm_model.predict(X_val_scaled)
svm_train_accuracy = accuracy_score(y_train, y_train_pred_svm)
svm_val_accuracy = accuracy_score(y_val, y_val_pred_svm)
print(f"SVM Training Accuracy: {svm_train_accuracy:.2f}")
print(f"SVM Validation Accuracy: {svm_val_accuracy:.2f}")
```

```
SVM Training Accuracy: 0.80
SVM Validation Accuracy: 0.80
```

```
y_test_pred_svm = svm_model.predict(X_test_scaled)

svm_test_accuracy = accuracy_score(y_test, y_test_pred_svm)
svm_precision = precision_score(y_test, y_test_pred_svm, average='weighted')
svm_recall = recall_score(y_test, y_test_pred_svm, average='weighted')
svm_f1 = f1_score(y_test, y_test_pred_svm, average='weighted')

print(f"SVM Test Accuracy: {svm_test_accuracy:.2f}")
print(f"Precision: {svm_precision:.2f}")
print(f"Recall: {svm_recall:.2f}")
print(f"F1 Score: {svm_f1:.2f}")
```

```
SVM Test Accuracy: 0.80
Precision: 0.80
Recall: 0.80
F1 Score: 0.80
```

Figure 12 – SVM Validation and Testing

```
tn, fp, fn, tp = conf_matrix_svm.ravel()

# Specificity
specificity = tn / (tn + fp)

# Balanced accuracy
balanced_acc = balanced_accuracy_score(y_test, y_test_pred_svm)

print(f"Specificity: {specificity:.2f}")
print(f"Balanced Accuracy: {balanced_acc:.2f}")
```

```
Specificity: 0.81
Balanced Accuracy: 0.80
```

Figure 13 – Balanced accuracy for SVM

4.8.2 Multilayer Perceptron

Train the MLP model with hidden layers 100 and 50.

```
print("Training MLP...")
mlp_model = MLPClassifier(hidden_layer_sizes=(100, 50), max_iter=300, random_state=42)
mlp_model.fit(X_train_scaled, y_train)
```

Figure 14 – MLP Training

MLP is then evaluated with the metrics shown in below image.

```
y_train_pred_mlp = mlp_model.predict(X_train_scaled)
y_val_pred_mlp = mlp_model.predict(X_val_scaled)

mlp_train_accuracy = accuracy_score(y_train, y_train_pred_mlp)
mlp_val_accuracy = accuracy_score(y_val, y_val_pred_mlp)

print(f"MLP Training Accuracy: {mlp_train_accuracy:.2f}")
print(f"MLP Validation Accuracy: {mlp_val_accuracy:.2f}")
```

```
MLP Training Accuracy: 1.00
MLP Validation Accuracy: 0.97
```

```
y_test_pred_mlp = mlp_model.predict(X_test_scaled)

mlp_test_accuracy = accuracy_score(y_test, y_test_pred_mlp)
mlp_precision = precision_score(y_test, y_test_pred_mlp, pos_label="yes")
mlp_recall = recall_score(y_test, y_test_pred_mlp, pos_label="yes")
mlp_f1 = f1_score(y_test, y_test_pred_mlp, pos_label="yes")

print(f"MLP Test Accuracy: {mlp_test_accuracy:.2f}")
print(f"Precision: {mlp_precision:.2f}")
print(f"Recall: {mlp_recall:.2f}")
print(f"F1 Score: {mlp_f1:.2f}")
```

```
MLP Test Accuracy: 0.98
Precision: 0.96
Recall: 0.99
F1 Score: 0.98
```

Figure 15 – Evaluation of MLP

4.8.3 ResNet-34

Load the spectrogram images and transform with respect to ResNet-34. Split the loaded and transformed dataset into train, validation and test data

```
image_dir = r"D:\Ashraf\NCI\Semester 3\Code\balanced_spectrogram_images"

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])

dataset = datasets.ImageFolder(root=image_dir, transform=transform)

train_size = int(0.8 * len(dataset))
val_size = int(0.1 * len(dataset))
test_size = len(dataset) - train_size - val_size

train_dataset, val_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, val_size, test_size])

train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)
test_loader = DataLoader(test_dataset, batch_size=32, shuffle=False)
```

Figure 16 – Load spectrogram data

Train the ResNet-34 model.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
resnet_model = models.resnet34(pretrained=True)

resnet_model.fc = nn.Linear(resnet_model.fc.in_features, 2)
resnet_model = resnet_model.to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(resnet_model.parameters(), lr=0.001)

epochs = 10
best_val_accuracy = 0.0

# Training Phase
for epoch in range(epochs):
    resnet_model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = resnet_model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)
```

```python
# Training Phase
for epoch in range(epochs):
    resnet_model.train()
    running_loss = 0.0
    correct = 0
    total = 0

    for images, labels in train_loader:
        images, labels = images.to(device), labels.to(device)

        optimizer.zero_grad()
        outputs = resnet_model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()
        _, preds = torch.max(outputs, 1)
        correct += (preds == labels).sum().item()
        total += labels.size(0)

    train_accuracy = correct / total

    # Validation Phase
    resnet_model.eval()
    val_correct = 0
    val_total = 0
    val_loss = 0.0
    with torch.no_grad():
        for images, labels in val_loader:
            images, labels = images.to(device), labels.to(device)
            outputs = resnet_model(images)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
            _, preds = torch.max(outputs, 1)
            val_correct += (preds == labels).sum().item()
            val_total += labels.size(0)

    val_accuracy = val_correct / val_total

    if val_accuracy > best_val_accuracy:
        best_val_accuracy = val_accuracy
        torch.save(resnet_model.state_dict(), "resnet34_best.pth")

    print(f"Epoch {epoch+1}/{epochs} - Train Loss: {running_loss/len(train_loader):.4f}, "
          f"Train Accuracy: {train_accuracy*100:.2f}% | Val Loss: {val_loss/len(val_loader):.4f}, "
          f"Val Accuracy: {val_accuracy*100:.2f}%")

print("Training complete. Best validation accuracy:", best_val_accuracy)
```

Figure 17 – Training of ResNet-34

```python
test_accuracy = accuracy_score(true_labels_test, predicted_labels_test)
test_precision = precision_score(true_labels_test, predicted_labels_test, average="weighted")
test_recall = recall_score(true_labels_test, predicted_labels_test, average="weighted")
test_f1 = f1_score(true_labels_test, predicted_labels_test, average="weighted")

print(f"Test Accuracy: {test_accuracy:.2f}")
print(f"Test Precision: {test_precision:.2f}")
print(f"Test Recall: {test_recall:.2f}")
print(f"Test F1 Score: {test_f1:.2f}")
```

```
Test Accuracy: 0.94
Test Precision: 0.94
Test Recall: 0.94
Test F1 Score: 0.94
```

Figure 18 – Evaluation of ResNet-34