Федеральное государственное бюджетное
образовательное учреждение высшего образования
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра прикладной математики

Курсовой проект по курсу

«МЕТОДЫ КОНЕЧНЫХ ЭЛЕМЕНТОВ»

Группа ПМ-12

Студент КУСАКИН АЛЕКСАДР

Новосибирск

2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Технологии распознавания и идентификации голоса становятся важной составляющей множества областей, включая безопасность, аутентификацию пользователей и интерактивные голосовые системы. Благодаря быстрому развитию вычислительных мощностей и появлению новых алгоритмов, методы, основанные на использовании нейросетей, приобретают значительную популярность и демонстрируют высокую эффективность при решении задач идентификации личности по голосу.

Основная цель данного исследования заключается в изучении существующих нейросетевых подходов и их применении для решения задачи идентификации голоса. В ходе работы будут рассмотрены существующие архитектуры нейронных сетей, такие как рекуррентные нейронные сети (RNN), свёрточные нейронные сети (CNN), а также модели на основе трансформеров. Будет проведён сравнительный анализ их эффективности в контексте задач идентификации.

Задачи исследования включают в себя изучение теоретической базы распознавания голосовых сигналов, сбор и подготовку соответствующих датасетов, а также разработку и обучение моделей нейронных сетей для последующего анализа их работы.

Актуальность темы обусловлена высоким спросом на надёжные и точные системы идентификации голоса, особенно в свете возросших требований к информационной безопасности. Результаты данного исследования могут быть полезны для разработки более совершенных систем аутентификации и повышения их устойчивости к различным видам атак.

.

1. ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ ИДЕНТИФИКАЦИИ ПО ГОЛОСУ

Методы идентификации по голосу можно разделить на три основные категории: традиционные, методы на основе машинного обучения и нейросетевые подходы.

1.1. ТРАДИЦИОННЫЕ МЕТОДЫ

Линейное предсказание кодирования (LPC): эта техника используется для моделирования вокального тракта и извлечения важной информации из голосового сигнала.

Метод спектрального анализа: включает такие техники, как анализ спектрограммы, мел-частотные кепстральные коэффициенты (MFCC), которые позволяют извлекать характеристики, устойчивые к изменчивости голоса.

1.2. КЛАССИЧЕСКИЕ МЕТОДЫ МАШИННОГО ОБУЧЕНИЯ

Гауссовские смесевые модели (GMM): применяются для моделирования вероятностного распределения признаков голосовых данных.

Поддерживающие векторные машины (SVM): эффективны для бинарной классификации голосовых данных на основе извлечённых признаков.

1.3. НЕЙРОСЕТЕВЫЕ МЕТОДЫ

Рекуррентные нейронные сети (RNN): обладают способностью сохранять информацию о последовательности и эффективно использовать её для анализа временных зависимостей в голосе.

Свёрточные нейронные сети (CNN): используются для изучения локальных паттернов и извлечения пространственных признаков из данных.

Трансформеры: современные архитектуры, такие как BERT и Transformer, благодаря своей способности учитывать контекст и предыдущее состояние, показывают высокую эффективность в задаче идентификации по голосу.

В дальнейшем будем рассматривать именно эти методы.

2. НЕЙРОСЕТЕВЫЕ ПОДХОДЫ К ИДЕНТИФИКАЦИИ ГОЛОСА

2.1. РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ (RNN)

Рекуррентные нейронные сети, благодаря своей архитектуре, способной работать с последовательными данными, широко используются в задачах обработки речи. Их основное преимущество — возможность сохранения информации о предыдущих состояниях, что делает их подходящими для анализа временных зависимостей в голосовом сигнале.

2.2. ДОЛГОСРОЧНАЯ КРАТКОВРЕМЕННАЯ ПАМЯТЬ (LSTM)

LSTM-модели — это усовершенствованный тип RNN, которые справляются с проблемами исчезающего градиента, что позволяет им эффективно запоминать информацию на более длительные временные промежутки. Это делает их особенно полезными в задачах, где требуется долгосрочное запоминание последовательности данных.

2.3. СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ (CNN)

Хотя CNN традиционно используются в задачах, связанных с изображениями, они также находят применение в обработке речевых данных, извлекая пространственные признаки из спектрограмм. Они хорошо справляются с задачей идентификации ключевых паттернов в голосе, что может улучшить конечную точность системы. В данной работе будет приведён пример именно CNN, ввиду ее универсальности для разных задач

2.4. ТРАНСФОРМЕРЫ

Трансформеры представляют собой современный подход, способный анализировать большие последовательности данных, при этом учитывая контекст

каждой части. Они обеспечивают высокую точность и часто использовали для моделей обработки естественного языка и распознавания речи.

2.5. КОМБИНИРОВАННЫЕ АРХИТЕКТУРЫ

Комбинация различных видов нейронных сетей позволяет извлечь и использовать преимущества каждой архитектуры. Например, модели, которые объединяют CNN и RNN, могут одновременно извлекать пространственные и временные признаки из голосовых данных, обеспечивая более высокое качество идентификации.

3. СВЁРТОЧНЫЕ НЕЙРОННЫЕ СЕТИ ДЛЯ ИДЕНТИФИКАЦИИ ГОЛОСА

Свёрточные нейронные сети изначально были разработаны для обработки двумерных данных, таких как изображения, но они также показали отличные результаты в задачах, связанных с обработкой аудиоданных, включая идентификацию голоса. В данном разделе будет рассмотрена архитектура CNN, подготовка данных и её применение в идентификации голоса.

3.1. ПОДГОТОВКА ДАННЫХ

Человеческий голос – это не одинокая волна, это сумма множества отдельных частот, создаваемых голосовыми связками, а также их гармоники. Из-за этого в обработке сырых данных волны тяжело найти закономерности голоса.

Нам на помощь придет преобразование Фурье – математический способ описать одну сложную звуковую волну спектрограммой, то есть набором множества частот и амплитуд. Эта спектрограмма содержит всю ключевую информацию о звуке: так мы узнаем, какие в исходном голосе содержатся частоты.

Но преобразование Фурье – математическая функция, которая нацелена на идеальный, неменяющийся звуковой сигнал, поэтому она требует практической адаптации. Так что, вместо того чтобы выделять частоты из всей записи сразу, эту запись мы поделим на небольшие отрезки, в течение которых звук не будет меняться. И применим преобразование к каждому из кусочков. Длительность блока: в среднем один слог человек произносит за 70 - 80 мс, а интонационно выделенный вдвое дольше – 100 - 150 мс [1]

Следующий шаг – посчитать спектрограмму второго порядка. Это нужно сделать, поскольку спектрограмма, помимо основных частот, также содержит гармоники, которые не очень удобны для анализа: они дублируют информацию. Расположены эти гармоники на равном друг от друга расстоянии, единственное их различие – уменьшение амплитуды.

Посмотрим, как выглядит спектр монотонного звука. Начнем с волны – синусоиды (рисунок 3.1.1), которую издает, например, проводной телефон при наборе номера.

Рисунок 3.1.1 – Преобразование Фурье от синуса

Видно, что, кроме основного пика, на самом деле представляющего сигнал, есть меньшие пики, гармоники, которые полезной информации не несут. Именно поэтому, прежде чем получать спектрограмму второго порядка, первую спектрограмму логарифмируют (рисунок 3.1.2).

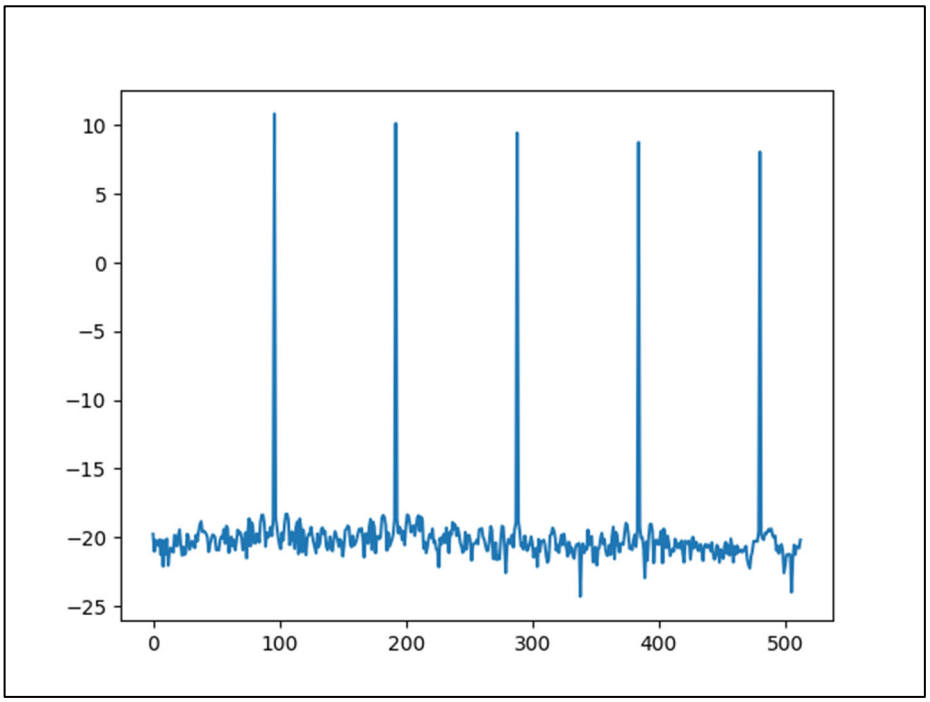

Рисунок 3.1.2 – Логарифм спектрограммы синуса

Теперь, если мы будем искать спектрограмму второго порядка – кепстр (рисунок 3.1.3), мы получим во много раз более приличную картинку, которая полностью, одним пиком, отображает нашу изначальную монотонную волну.

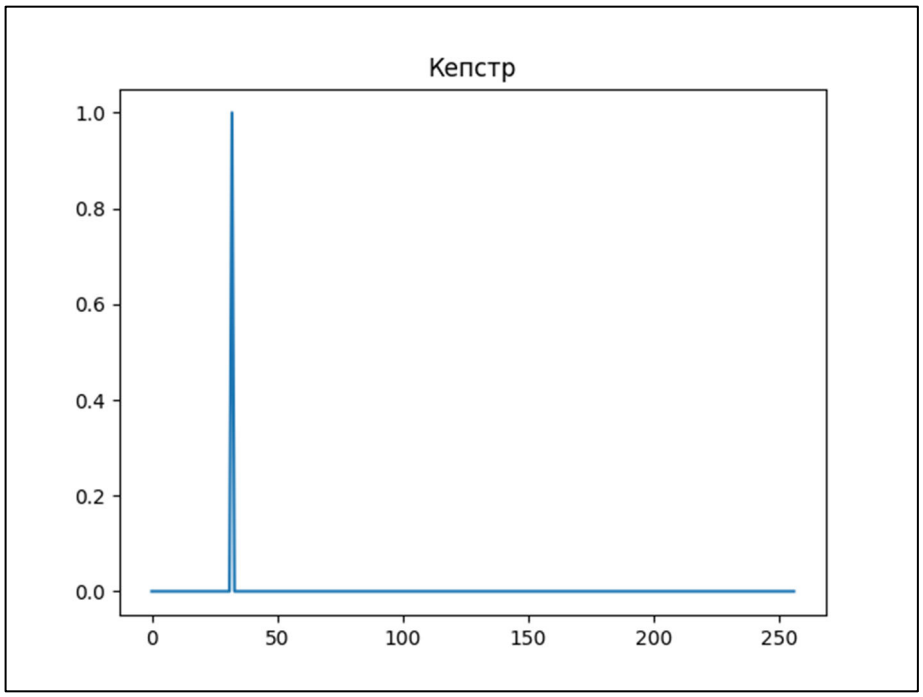

Рисунок 3.1.3 – Кепстр

Из-за нелинейности слуха к восприятию частот можно вывести закономерность – *мел*.

$$m = 1127 \cdot \ln\left(1 + \frac{f}{100}\right) \quad (3.1.1)$$

– зависимость мела от герца.

3.2. ОСНОВНЫЕ КОМПОНЕНТЫ CNN

CNN состоят из нескольких ключевых компонентов, каждый из которых играет важную роль в процессе извлечения признаков:

- **Свёрточные слои:** это основа сети, где применяются фильтры (или ядра) для извлечения локальных признаков из входных данных. Каждый фильтр скользит по спектрограмме, создавая карту признаков, отражающую присутствие различных шаблонов.
- **Слои объединения:** обычно после свёрточных слоёв следует слой объединения, который уменьшает размер карты признаков, сохраняя при этом наиболее значимые элементы. Наиболее распространённый тип — это максимальное объединение, которое выбирает максимальное значение в окне фильтра.
- **Слои активации:** после каждого свёрточного слоя часто используется функция активации, такая как ReLU, чтобы ввести в модель нелинейность, повышая её способность моделировать сложные зависимости в данных.
- **Полносвязные слои:** в конце сети присутствуют полносвязные слои, которые соединяют детектированные признаки для принятия решений.

3.3. ПРИМЕНЕНИЕ CNN К ИДЕНТИФИКАЦИИ ГОЛОСА

Для обработки аудиоданных с использованием CNN, звуковые сигналы сначала переводятся в спектрограммы или другие двумерные представления, такие как мел-спектрограммы (рисунок 3.3.1). Эти визуализированные данные затем

подаются на вход CNN, где свёрточные слои извлекают ключевые пространственные признаки, необходимые для идентификации.

- **Извлечение признаков:** свёрточные слои обучаются выделять характерные признаки голосового сигнала, такие как форманты и качество голоса.
- **Классификация:** полносвязные слои сети принимают детектированные признаки и классифицируют их, определяя, к какому голосу (или спикеру) они принадлежат.

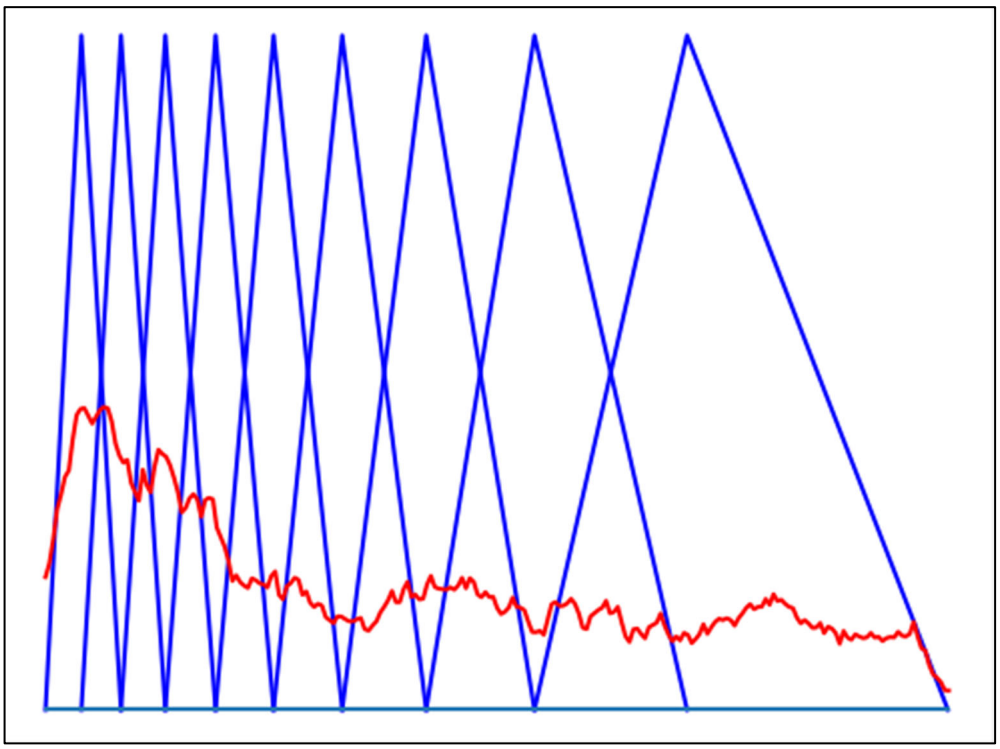

Рисунок 3.3.1 – Вычисление мел-частотных кепстральных коэффициентов

4. ОБУЧЕНИЕ CNN

Возьмем готовую одиннадцатислойную нейросеть и обучим ее на датесете voxceleb с голосами актеров.

4.1. ЗАГРУЗКА ДАТАСЕТА

Датасет представлен в виде списка спикеров с url и тайм-кодами отрывков голов. Для его загрузки используем программу из приложения 1, которая скачивает видео, затем редерит его в необходимом формате (он указан в инструкции к датасету), далее нарезает, и расфасовывает по директориям, где название директории – это имя спикера(класса). Затем конвертирует из .mp4 в .wav.

Ввиду того, что датасет состоит из тяжелых файлов, к тому же часть видео недоступна по тем или иным причинам, было принято решение взять 5 человек и на каждого по 100 фрагментов: 80 тренировка и 20 тестирование.

4.2. ОПРЕДЕЛЕНИЕ МОДЕЛИ

Ниже представлен код одиннадцатислойной CNN, где 4 свёрточных слоя, 4 слоя пулинга и 3 полносвязных слоя.

```
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1,
padding=1)  # Свёрточный слой 1
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)  # Свёрточный слой 2
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1,
padding=1)  # Свёрточный слой 3
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=1,
padding=1)  # Свёрточный слой 4
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)  #
Слой pooling
```

```
        self.fc1 = None
        self.fc2 = None
        self.fc3 = None
        self.num_classes = num_classes

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # Свёрточный слой 1 и pooling
        x = self.pool(F.relu(self.conv2(x)))  # Свёрточный слой 2 и pooling
        x = self.pool(F.relu(self.conv3(x)))  # Свёрточный слой 3 и pooling
        x = self.pool(F.relu(self.conv4(x)))  # Свёрточный слой 4 и pooling

        if self.fc1 is None or self.fc2 is None or self.fc3 is None:
            num_features = x.view(x.size(0), -1).size(1)  # Подготовка
к полносвязным слоям
            self.fc1 = nn.Linear(num_features, 256).to(device)  #
Полносвязный слой 1
            self.fc2 = nn.Linear(256, 128).to(device)  # Полносвязный
слой 2
            self.fc3 = nn.Linear(128, self.num_classes).to(device)  #
Полносвязный выходной слой

        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))  # Полносвязный слой 1 с активацией ReLU
        x = F.relu(self.fc2(x))  # Полносвязный слой 2 с активацией ReLU
        x = self.fc3(x)  # Полносвязный выходной слой
        return x
```

4.3. РЕЗУЛЬТАТ ОБУЧЕНИЯ

После обучение 50 эпох, на тестовый данных получился результат ниже.

```
Accuracy of the model on the test data: 76.47058823529412%
```

Результат оказался приемлемым, ввиду размера тренировочного датасета.

ЗАКЛЮЧЕНИЕ

В ходе данного исследования были рассмотрены различные методы идентификации по голосу, начиная от традиционных подходов и заканчивая современными методами машинного обучения, включая нейросетевые технологии. Основное внимание было уделено архитектурам свёрточных нейронных сетей (CNN).

Реализация CNN для задачи идентификации голоса позволила продемонстрировать их способность автоматически извлекать и анализировать ключевые пространственные признаки из спектрограмм, что значительно упрощает процесс подготовки данных и снижает зависимость от ручной инженерии признаков.

Однако, как и другие нейросетевые подходы, CNN предъявляют высокие требования к объёмам обучающих данных и вычислительным ресурсам. Это создаёт вызовы, связанные с необходимостью оптимизации моделей для работы в условиях ограниченных ресурсов, а также с поиском эффективных методов аугментации данных и выбора гиперпараметров.

Итоги работы подчеркивают значимость машинного обучения и нейросетевых технологий в современном мире, где потребность в надежных и высокоточных системах идентификации голоса продолжает расти.

СПИСОК ЛИТЕРАТУРЫ

1. Jerry Feldman NTL / Jerry Feldman [Электронный ресурс] // ICSI : [сайт]. — URL: https://www.icsi.berkeley.edu/icsi/projects/ai/ntl (дата обращения: 20.03.2025).
2. Бишоп К. М. Нейронные сети для распознавания образов / К. М. Бишоп. — Москва : Изд-во ДМК Пресс, 2020. — 736 с. — ISBN 978-5-94074-144-2.
3. Николенко С. Кадурин А. Архангельская Е. Глубокое обучение погружение в мир нейронных сетей [Текст] / Николенко С. Кадурин А. Архангельская Е. — 1-е изд. — СПб: ООО Издательство "Питер", 2018 — 481с.
4. Гудфеллоу Я. Глубокое обучение / Я. Гудфеллоу, И. Бенджио, А. Курвилль; пер. с англ. — Москва : Изд-во "Вильямс", 2018. — 720 с. — (ISBN 978-5-8459-2036-3).
5. Вологдин Э.И. Методы и алгоритмы обработки голосовых сигналов / Вологдин Э.И. [Электронный ресурс] // soundmain.ru : [сайт]. — URL: https://soundmain.ru/resources/vologdin-eh-i-metody-i-algoritmy-obrabotki-zvukovyx-signalov.294/ (дата обращения: 16.03.2025).
6. Huang X., Deng L. Deep Learning for Speech and Language Processing / X. Huang, L. Deng. — [Электронный ресурс]. — URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/ICASSP-2013-OverviewMSRDeepLearning.pdf (дата обращения: 22.03.2025).

ПРИЛОЖЕНИЯ

ПРИЛОЖЕНИЕ 1: ЗАГРУЗКА ДАТАСЕТА

```
# %%
import pandas as pd
import subprocess
import os
from pydub import AudioSegment

# %%
def log(message):
    print(f"[LOG] {message}")

# %%
class Segment:
    def __init__(self, name, start, end):
        self.name = name
        self.start = start
        self.end = end

# %%
class Data:
    def __init__(self, uri, speaker, verification, identification,
segments=None):
        self.segments = segments if segments is not None else []
        self.uri = uri
        self.speaker = speaker
        self.verification = verification
        self.identification = identification

    def AddSegment(self, segment):
        self.segments.append(segment)

    def Print(self):
        print("SPEAKER: ", self.speaker)
        print("URI: ", self.uri)
        print("SEGMENTS: ")
        for segment in self.segments:
            print('\t name: ', segment.name)
            print('\t start: ', segment.start)
            print('\t end: ', segment.end)

# %%
def DecodeFromCSV(csv_file):
    data = pd.read_csv(csv_file)

    data_objects = {}

    for index, row in data.iterrows():
        segment = Segment(row['segment'], row['start'], row['end'])
        uri = row['uri']

        if uri not in data_objects:
```

```
            data_objects[uri] = Data(uri, row['speaker'],
row['verification'], row['identification'])

        data_objects[uri].AddSegment(segment)

    return list(data_objects.values())

# %%
data_objects =
DecodeFromCSV('./datasets/voxceleb/data/v1/voxceleb1.csv')

# %%
def download_youtube_video(youtube_id,
save_directory='./data_video/temp'):
    output_path = os.path.join(save_directory, f'{youtube_id}.mp4')
    if not os.path.exists(output_path):
        log(f"Скачивание видео {youtube_id}...")
        command = [
            'yt-dlp',
            '-f', 'bestvideo+bestaudio',
            '--merge-output-format', 'mp4',
            f'https://www.youtube.com/watch?v={youtube_id}',
            '--output', output_path
        ]
        process = subprocess.Popen(command, stdout=subprocess.PIPE,
stderr=subprocess.STDOUT, text=True)

        for output in process.stdout:
            print(output, end='')

        log(f"Видео {youtube_id} скачано!")

# %%
def convert_video_gpu(input_file, output_file):
    log(f"Конвертация видео {input_file} в {output_file} с
использованием GPU...")
    command = [
        'ffmpeg', '-threads', '1', '-y', '-hwaccel', 'cuda', '-i',
input_file,
        '-async', '1', '-qscale:v', '5', '-r', '25',
        '-vf', 'yadif,scale=640:360', '-c:v', 'h264_nvenc', '-c:a',
'aac',
        output_file
    ]
    result = subprocess.run(command, text=True, capture_output=True)

    if result.returncode != 0:
        log(f"Ошибка при конвертации {input_file}: {result.stderr}")
        with open("error_log.txt", "a") as log_file:
            log_file.write(f"Ошибка с {input_file}: {result.stderr}\n")
    else:
        log(f"Видео {input_file} сконвертировано в {output_file}!")

# %%
def convert_video_cpu(input_file, output_file):
    log(f"Конвертация видео {input_file} в {output_file}...")
    command = [
        'ffmpeg', '-threads', '1', '-y', '-i', input_file,
```

```
        '-async', '1', '-qscale:v', '5', '-r', '25',
        '-vf', 'yadif,scale=trunc(iw/2)*2:-1', '-c:v', 'libx264', '-
c:a', 'aac',
        output_file
    ]
    result = subprocess.run(command, text=True, capture_output=True)

    if result.returncode != 0:
        log(f"Ошибка при конвертации {input_file}: {result.stderr}")
        with open("error_log.txt", "a") as log_file:
            log_file.write(f"Ошибка с {input_file}: {result.stderr}\n")
    else:
        log(f"Видео {input_file} сконвертировано в {output_file}!")

# %%
def slice_video(input_file, start_time, end_time, output_file):
    log(f"Нарезка видео {input_file} с {start_time} до {end_time} в
{output_file}...")

    os.makedirs(os.path.dirname(output_file), exist_ok=True)

    command = [
        'ffmpeg', '-y', '-i', input_file, '-ss', str(start_time),
        '-to', str(end_time), '-c:v', 'libx264', '-c:a', 'aac',
output_file
    ]
    result = subprocess.run(command, text=True, capture_output=True)

    if result.returncode != 0:
        log(f"Ошибка при нарезке {input_file}: {result.stderr}")
        with open("error_log.txt", "a") as log_file:
            log_file.write(f"Ошибка с {input_file}: {result.stderr}\n")
    else:
        log(f"Фрагмент сохранен в {output_file}!")

# %%
MAX_VIDEO_BY_PERSON = 100
MAX_PERSON = 5

# %%
counter_video_map = {}
counter_video = 0
counter_person = 0

# %%
missing_videos_log = './missing_videos.txt'
for data in data_objects:

    if counter_person >= MAX_PERSON + 1:
        break

    if data.speaker not in counter_video_map:
        counter_video_map[data.speaker] = 0
        counter_person += 1
    counter_video = counter_video_map[data.speaker]
    if counter_video >= MAX_VIDEO_BY_PERSON:
        continue
```

```
    print('PERSON: ',  data.speaker, 'CURRENT VIDEO: ', counter_video)

    youtube_id = data.uri.split('/')[-1]
    input_file = f'./data_video/temp/{youtube_id}.mp4'
    converted_file = f'./data_video/temp/converted_{youtube_id}.mp4'

    if not os.path.exists(input_file):
        download_youtube_video(youtube_id)

    if os.path.exists(input_file):

        if not os.path.exists(converted_file):
            convert_video_gpu(input_file, converted_file)

        if os.path.exists(converted_file):
            for segment in data.segments:
                if counter_video < MAX_VIDEO_BY_PERSON:
                    log("Segmentation is started")
                    start_time = segment.start
                    end_time = segment.end
                    output_file = f'./data_video/{segment.name}.mp4'
                    slice_video(converted_file, start_time, end_time,
output_file)
                    log("Segmentation is ended")
                    counter_video += 1
                else:
                    counter_video_map[data.speaker] = counter_video
                    break
    else:
        log(f"Файл {input_file} не найден, пропуск...")
        with open(missing_videos_log, 'a') as log_file:
            log_file.write(f"{youtube_id}\n")

    counter_video_map[data.speaker] = counter_video


# %%

def extract_and_split_audio(source_dir, train_dir, test_dir):
    for target_dir in [train_dir, test_dir]:
        if not os.path.exists(target_dir):
            os.makedirs(target_dir)

    for person_name in os.listdir(source_dir):
        person_path = os.path.join(source_dir, person_name)

        if os.path.isdir(person_path):
            files = [f for f in os.listdir(person_path) if
f.endswith('.mp4')]
            train_index = int(len(files) * 0.8)

            train_files = files[:train_index]
            test_files = files[train_index:]

            process_files(person_name, person_path, train_files,
train_dir)
            process_files(person_name, person_path, test_files,
test_dir)
```

```
def process_files(person_name, person_path, files, target_base_dir):
    target_person_path = os.path.join(target_base_dir, person_name)
    if not os.path.exists(target_person_path):
        os.makedirs(target_person_path)

    for file_name in files:
        video_path = os.path.join(person_path, file_name)
        audio_file_name = os.path.splitext(file_name)[0] + '.wav'
        audio_path = os.path.join(target_person_path, audio_file_name)

        video = AudioSegment.from_file(video_path, format='mp4')
        video.export(audio_path, format='wav')

source_directory = 'data_video'
train_directory = 'data_audio_train'
test_directory = 'data_audio_test'

extract_and_split_audio(source_directory, train_directory,
test_directory)
```

ПРИЛОЖЕНИЕ 2: ОБУЧЕНИЕ МОДЕЛИ

```
# %%
import os
import librosa
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from sklearn.metrics import classification_report, confusion_matrix

# %%
# Определение устройства
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f'Using device: {device}')

# %%
# Определение класса датасета
class AudioDataset(Dataset):
    def __init__(self, root_dir, sr=16000, n_mfcc=40, max_len=220):
        self.file_paths = []  # список путей к аудиофайлам
        self.labels = []      # соответствующие метки классов
        self.sr = sr
        self.n_mfcc = n_mfcc
        self.max_len = max_len  # максимальная длина для паддинга
        self._load_dataset(root_dir)

    def _load_dataset(self, root_dir):
```

```
        class_labels = os.listdir(root_dir)
        for label_idx, class_label in enumerate(class_labels):
            class_dir = os.path.join(root_dir, class_label)
            if os.path.isdir(class_dir):
                for file_name in os.listdir(class_dir):
                    if file_name.endswith('.wav'):
                        file_path = os.path.join(class_dir, file_name)
                        self.file_paths.append(file_path)
                        self.labels.append(label_idx)

    def __len__(self):
        return len(self.file_paths)

    def __getitem__(self, idx):
        file_path = self.file_paths[idx]
        label = self.labels[idx]
        audio, sr = librosa.load(file_path, sr=self.sr)
        mfcc = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=self.n_mfcc)
        mfcc = (mfcc - np.mean(mfcc)) / np.std(mfcc)

        # Паддинг или обрезка
        if mfcc.shape[1] < self.max_len:
            pad_width = self.max_len - mfcc.shape[1]
            mfcc = F.pad(torch.tensor(mfcc, dtype=torch.float32), (0,
pad_width))
        else:
            mfcc = torch.tensor(mfcc, dtype=torch.float32)[:,
:self.max_len]

        mfcc = mfcc.unsqueeze(0)
        return mfcc, label

# %%
# Определение модели
class CNN(nn.Module):
    def __init__(self, num_classes):
        super(CNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1,
padding=1)  # Свёрточный слой 1
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)  # Свёрточный слой 2
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, stride=1,
padding=1)  # Свёрточный слой 3
        self.conv4 = nn.Conv2d(128, 256, kernel_size=3, stride=1,
padding=1)  # Свёрточный слой 4
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2, padding=0)  #
Слой pooling

        self.fc1 = None
        self.fc2 = None
        self.fc3 = None
        self.num_classes = num_classes

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))  # Свёрточный слой 1 и
pooling
        x = self.pool(F.relu(self.conv2(x)))  # Свёрточный слой 2 и
pooling
```

```
        x = self.pool(F.relu(self.conv3(x)))  # Свёрточный слой 3 и
pooling
        x = self.pool(F.relu(self.conv4(x)))  # Свёрточный слой 4 и
pooling

        if self.fc1 is None or self.fc2 is None or self.fc3 is None:
            num_features = x.view(x.size(0), -1).size(1)  # Подготовка
к полносвязным слоям
            self.fc1 = nn.Linear(num_features, 256).to(device)  #
Полносвязный слой 1
            self.fc2 = nn.Linear(256, 128).to(device)  # Полносвязный
слой 2
            self.fc3 = nn.Linear(128, self.num_classes).to(device)  #
Полносвязный выходной слой

        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))  # Полносвязный слой 1 с активацией
ReLU
        x = F.relu(self.fc2(x))  # Полносвязный слой 2 с активацией
ReLU
        x = self.fc3(x)  # Полносвязный выходной слой
        return x

# %%
# Использование
dataset = AudioDataset(root_dir='data_audio_train')
dataloader = DataLoader(dataset, batch_size=500, shuffle=True)

num_classes = len(set(dataset.labels))
model = CNN(num_classes).to(device)

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

# %%
# Тренировка модели
for epoch in range(50):
    for inputs, targets in dataloader:
        inputs, targets = inputs.to(device), targets.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, targets)
        loss.backward()
        optimizer.step()
    print(f'Epoch {epoch+1}, Loss: {loss.item()}')

# %%
# Создание тестового датасета
test_dataset = AudioDataset(root_dir='data_audio_test')
test_dataloader = DataLoader(test_dataset, batch_size=100,
shuffle=False)

model.eval()
correct = 0
total = 0

# %%
# Тестирование
```

```
with torch.no_grad():
    for inputs, labels in test_dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)

        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy of the model on the test data: {accuracy}%')

# %%
# Собираем все истинные и предсказанные метки
true_labels = []
pred_labels = []

with torch.no_grad():
    for inputs, labels in test_dataloader:
        inputs, labels = inputs.to(device), labels.to(device)
        outputs = model(inputs)
        _, predicted = torch.max(outputs.data, 1)

        true_labels.extend(labels.cpu().numpy())
        pred_labels.extend(predicted.cpu().numpy())

print("Classification Report:")
print(classification_report(true_labels, pred_labels))

print("Confusion Matrix:")
print(confusion_matrix(true_labels, pred_labels))
```