

Руководство разработчика

1 Основные сведения

1.1 Назначение и условия применения программы

Разработанная открытая библиотека PyAra может быть использована при создании следующих информационных систем:

- Системы голосовой аутентификации (call-центры, программное обеспечение банкоматов, системы «умный дом»).
- Системы проверки аудиопотока для выявления злонамеренных попыток компрометации выступающего (публичного должностного лица).
- Системы расследования инцидентов, связанных с обманом (фродом) физических лиц мошенниками с использованием звонков по телефону или через популярные мессенджеры.
- Системы предотвращения мошенничества в мессенджерах и социальных сетях, содержащих функции голосового общения.
- Индивидуальные помощники (т.н. умные колонки), воспринимающие голосовые команды (покупки, оплата счетов, банковские переводы).

Подобная система может либо стать частью подсистемы, либо самостоятельной обособленной системой защиты от спуфинг-атак посредством синтеза голоса.

Высокая точность определения системой синтезированной аудиозаписи дает возможность сделать ее первичным «фильтром», через который будут проходить все аудиозаписи. Это сильно облегчит дальнейшую обработку пропущенных через фильтр аудиозаписей.

1.2 Установка

Для успешной установки библиотеки PyAra необходимо выполнить следующую команду:

```
pip install Pyara
```

1.3 Зависимости и минимальные технические требования

Библиотека PyAra имеет следующие зависимости, которые будут установлены вместе с самой библиотекой, либо должны быть установлены заранее:

- torch \geq 1.13.1
- torchaudio \geq 0.13.1
- soundfile \geq 0.12.1
- numpy
- librosa

Минимальные технические требования для запуска и использования открытой библиотеки PyAra:

- 1) Версия Python $>$ 3.7.
- 2) Операционная система: Linux, Windows, MacOS.
- 3) Для запуска функций библиотеки необходимо как минимум 100 Мбайт оперативной памяти.

1.4 Характеристика программы

В библиотеку PyAra входит предварительно обученная модель, которая может быть использована по умолчанию. Данная модель обучена на русскоязычном датасете, который был разработан специально для этой библиотеки. С моделью по умолчанию PyAra умеет детектировать синтезированные русскоязычные аудиозаписи. Так же имеется возможность обучать и встраивать собственные модели, обученные даже на других языках. Или же улучшать уже имеющуюся модель.

Представленная модель достигает точности более 95% на собранном датасете. Тем самым данная модель показывает очень высокие результаты в детектировании поддельной речи и может быть встроена в различные системы.

Время работы библиотеки, а точнее ее основной функции детектирования синтезированной речи, довольно небольшое, на обычном пользовательском компьютере проверка одной аудиозаписи на подлинность

занимает около 50 мс. Данное время обычному пользователю будет почти незаметно, что означает, что данная библиотека за небольшие временные затраты может существенно помочь с проверкой входных аудиозаписей.

2 Описание функций

2.1 Функция `predict_audio`

Функция, `predict_audio`, предназначена для предсказания типа аудио (подлинное или синтезированное) с использованием заданной модели.

Входные данные:

- `file_path` — это путь до аудиофайла, тип которого мы хотим предсказать при помощи нейронной сети.
- `print_probability` — Флаг, указывающий, нужно ли выводить вероятность предсказания (по умолчанию установлен в `False`).
- `pitch_shift` — Значение смещения тональности аудио (по умолчанию равно 0).
- `sample_rate` — Частота дискретизации аудиосигнала.
- `width` — Ширина окна, используемая при обработке аудиосигнала.

Выходные данные: возвращает предсказание аудио в виде целого числа: 0, если аудио подлинное, 1, если аудио синтезированное.

Описание работы функции:

- Загружает модель для предсказания (вызывает функцию `model_eval`).
- Подготавливает аудиосигнал с помощью функции `prepare_signal`, принимая во внимание параметры, такие как смещение тональности, ширина окна и частота дискретизации;
- Передает подготовленный сигнал в модель для получения предсказания и вероятности;
- Возвращает предсказание аудио в виде целого числа: 0, если аудио подлинное, 1, если аудио синтезированное;
- Если установлен флаг `print_probability`, возвращает строку с ответом и вероятностью предсказания;
- Если флаг `print_probability` не установлен, возвращает только предсказание аудио.

Пример использования

```
from pyara import predict_audio  
predict_audio('C:/Tests/mp3.mp3')
```

2.2 Функция cut_if_necessary

Функция cut_if_necessary предназначена для обрезки аудиосигнала до заданной ширины (количества выборок) в случае, если исходный сигнал имеет большую длину.

Входные данные:

- signal (torch.Tensor): Аудиосигнал;
- width (int): Ширина сигнала после обрезки. По умолчанию CFG.width = 400.

Выходные данные: torch.Tensor: Обрезанный аудиосигнал.

Описание работы функции:

- Проверяет длину аудиосигнала (по оси времени, представленной третьим измерением тензора).
- Если длина аудиосигнала превышает заданную ширину (width), то происходит обрезка, удаляя лишние выборки.
- Возвращает обрезанный аудиосигнал.

Пример использования

```
from pyara import cut_if_necessary  
cut_if_necessary('C:/Tests/mp3.mp3')
```

2.3 Функция right_pad_if_necessary

Функция right_pad_if_necessary выполняет дополнение последнего измерения входного сигнала вправо, если необходимо.

Входные данные:

- signal (torch.Tensor): Тензор сигнала, представляющий многомерные данные. Должен быть трехмерным тензором с размерностью [batch_size, num_channels, signal_length].
- width (int): Желаемая ширина сигнала после дополнения (по умолчанию берется из CFG).

Выходные данные: torch.Tensor (дополненный сигнал с выполненным дополнением последнего измерения вправо).

Пример использования

```
from pyra import right_pad_if_necessary  
right_pad_if_necessary('C:/Tests/mp3.mp3')
```

2.4 Функция prepare_signal

Функция prepare_signal предназначена для подготовки аудиосигнала для дальнейшей обработки с использованием нейронной сети.

Входные данные:

- voice_path — Путь к аудиофайлу;
- pitch_shift — Значение смещения тональности аудио (по умолчанию равно 0);
- width — Ширина сигнала после обрезки;
- sample_rate — Частота дискретизации аудиосигнала.

Выходные данные: signal — подготовленный сигнал для обработки.

Описание работы функции:

- Загрузка аудиофайла с помощью функции torchaudio.load(voice_path). Возвращает сигнал signal и частоту дискретизации sample_rate.
- Усреднение сигнала signal по первому измерению (каналам) с использованием функции signal.mean(dim=0).
- Добавление размерности пакета (batch) к сигналу signal с помощью функции signal.unsqueeze(dim=0).
- Если задано смещение тональности (pitch_shift), применение эффекта сдвига тона с использованием функции librosa.effects.pitch_shift.
- Применение преобразования MFCC (Mel-frequency cepstral coefficients) к сигналу signal с помощью функции MFCC_spectrogram.
- Обрезка и дополнение спектрограммы, если необходимо, с использованием функций cut_if_necessary и right_pad_if_necessary.

- Повторение спектрограммы `signal` 3 раза по первому измерению (пакету) с использованием функции `signal.repeat(3, 1, 1)`.
- Добавление размерности пакета (`batch`) к спектрограмме `signal` с помощью функции `signal.unsqueeze(dim=0)`.
- Перемещение спектрограммы `signal` на устройство (например, GPU), указанное в `CFG.device`, с использованием функции `signal.to(CFG.device)`.
- Возврат подготовленного сигнала `signal`.

Пример использования

```
from pyara import prepare_signal  
prepare_signal('C:/Tests/mp3.mp3')
```

2.5 Функция `prepare_signals`

Функция `prepare_signals` предназначена для подготовки нескольких аудиосигналов для дальнейшей обработки с использованием нейронной сети.

Входные данные

- `voice_path` — Путь к аудиофайлу;
- `pitch_shift` — Значение смещения тональности аудио (по умолчанию равно 0);
- `width` — Ширина сигнала после обрезки;
- `sample_rate` — Частота дискретизации аудиосигнала.

Выходные данные: `signal` — Подготовленный сигнал для обработки.

Описание работы функции:

- Для каждого пути к аудиофайлу из списка `voice_paths` выполняются следующие шаги:
 - Загрузка аудиофайла с помощью функции `torchaudio.load(voice_path)`. Возвращает сигнал `signal` и частоту дискретизации `sample_rate`;
 - Усреднение сигнала `signal` по первому измерению (каналам) с использованием функции `signal.mean(dim=0)`;

- Добавление размерности пакета (batch) к сигналу signal с помощью функции `signal.unsqueeze(dim=0)`;
- Если задано смещение тональности (pitch_shift), применение эффекта сдвига тона с использованием функции `librosa.effects.pitch_shift`;
- Применение преобразования MFCC (Mel-frequency cepstral coefficients) к сигналу signal с помощью функции `MFCC_spectrogram`;
- Обрезка и дополнение спектрограммы, если необходимо, с использованием функций `cut_if_necessary` и `right_pad_if_necessary`;
- Повторение спектрограммы signal 3 раза по первому измерению (пакету) с использованием функции `signal.repeat(3, 1, 1)`;
- Добавление размерности пакета (batch) к спектрограмме signal с помощью функции `signal.unsqueeze(dim=0)`;
- Перемещение спектрограммы signal на устройство (например, GPU), указанное в `CFG.device`, с использованием функции `signal.to(CFG.device)`;
- Возврат подготовленного сигнала signal и добавление его в список res.

Пример использования

```
from pyara import prepare_signals  
prepare_signals(['C:/Tests/mp3.mp3',C:/Tests/mp23.mp3'] )
```

2.6 Функция prediction

Функция prediction выполняет предсказание с использованием заданной модели для входного сигнала.

Входные данные:

- model — Модель, которая будет использоваться для предсказания. Должна быть совместима с PyTorch и иметь метод forward для выполнения прямого прохода;
- signal — Входной сигнал, для которого будет выполнено предсказание. Должен быть трехмерным тензором с размерностью [batch_size, num_channels, signal_length];

- `print_probability` — Флаг, указывающий, нужно ли выводить вероятность предсказания (по умолчанию установлен в `False`).

Выходные данные: возвращает кортеж, содержащий предсказанную метку класса (0 или 1) и вероятность предсказания. Если `print_probability` установлен в `True`, также возвращает строку с ответом и вероятностью предсказания.

Описание работы функции:

- Переносит модель на устройство, указанное в `CFG.device`;
- Сжимает измерение сигнала с использованием `signal.squeeze()`;
- Выполняет прямой проход модели с входным сигналом, получая выходной тензор `output`;
- Применяет функцию `torch.nn.Softmax` для получения вероятностного распределения по классам;
- Возвращает метку класса и вероятность предсказания для класса 0 (реальный голос) и класса 1 (синтезированный голос);
- Если `print_probability` установлен в `True`, также выводит информацию о вероятности предсказания.

Пример использования

```
from pyara import prediction
model = model_eval()
signal = prepare_signal(file_path, pitch_shift, width, sample_rate)
prediction(model, signal)
```

2.7 Функция `prediction_multiple`

Функция `prediction_multiple` выполняет предсказание для нескольких входных сигналов с использованием заданной модели.

Входные данные:

- `model` — Модель, которая будет использоваться для предсказания. Должна быть совместима с PyTorch и иметь метод `forward` для выполнения прямого прохода;

- `signals` — Список входных сигналов, для которых будет выполнено предсказание. Каждый сигнал должен быть трехмерным тензором с размерностью `[batch_size, num_channels, signal_length]`.

Выходные данные: возвращает список, содержащий предсказанные метки классов для входных сигналов.

Каждая метка класса представлена значением 1, если модель предсказывает синтезированный голос, и 0, если модель предсказывает реальный голос.

Возвращает также список вероятностей предсказания для каждого сигнала.

Описание работы функции:

- Переносит модель на устройство, указанное в `CFG.device`.
- Для каждого входного сигнала из списка `signals` выполняет следующие шаги:

- 1) Сжимает измерение сигнала с использованием `signal.squeeze()`;
- 2) Выполняет прямой проход модели с входным сигналом, получая выходной тензор `output`;
- 3) Применяет функцию `torch.nn.Softmax` для получения вероятностного распределения по классам;
- 4) Определяет предсказанный класс с использованием `output.argmax(dim=-1).cpu().numpy()`;
- 5) Добавляет предсказанный класс и вероятность предсказания в соответствующие списки.

Пример использования

```
from pyara import prediction
model = model_eval()
signals=prepare_signals([file_path1,file_path2],pitch_shift,width,
sample_rate)
prediction_multiple(model, signals )
```

3 Примеры использования библиотеки

3.1 Инициализации модели и получения полной информации

```
from pyara.Model import model_eval
from torchsummary import summary
model = model_eval()
summary(model, input_size = (3,80,100))
```

Пример вывода информации о модели можно видеть на рисунке 1.

—ResNetBlock: 1-11	--
└─Conv2d: 2-56	9,248
└─BatchNorm2d: 2-57	64
└─LeakyReLU: 2-58	--
└─Dropout: 2-59	--
└─Conv2d: 2-60	9,248
└─Conv2d: 2-61	9,248
└─BatchNorm2d: 2-62	64
—LeakyReLU: 1-12	--
—BatchNorm2d: 1-13	64
—Dropout: 1-14	--
—LogSoftmax: 1-15	--
—Linear: 1-16	4,224
—Linear: 1-17	258
=====	
Total params: 255,650	
Trainable params: 255,650	
Non-trainable params: 0	
=====	

Рисунок 1 – Информация о параметрах модели

Команда `from torchsummary import summary` в Python используется для импорта функции `summary` из библиотеки `torchsummary`. Эта функция предоставляет удобный способ для получения краткого описания моделей в PyTorch. Вот основные моменты, которые она охватывает:

Функция `summary` позволяет быстро просмотреть все слои в модели PyTorch. Она отображает последовательность слоёв и их типы, что особенно полезно для проверки и отладки архитектуры модели.

`summary` предоставляет информацию о количестве обучаемых и необучаемых параметров в каждом слое и во всей модели. Это помогает понять, насколько сложной или тяжелой является модель.

Для каждого слоя отображаются размеры выходных данных. Это полезно для понимания, как изменяются размеры данных при прохождении через модель, и помогает выявлять потенциальные ошибки в размерах данных.

Функция оценивает объем памяти, который будет занимать каждый слой при выполнении прямого прохода. Это важно для оптимизации моделей, особенно при работе с ограниченными ресурсами, например, на мобильных устройствах или в веб-приложениях.

Чтобы использовать `summary`, необходимо сначала импортировать её, а затем вызвать, передав экземпляр модели и размеры входных данных, например: `summary(your_model, input_size=(channels, height, width))`. Это даст полное представление о модели, включая слои, параметры и расход памяти.

3.2 Пример работы с аудиосигналом

Для проверки работы функций библиотеки вы можете скачать аудио с нашего репозитория по ссылке https://github.com/efanov/pyara/tree/main/tests/test_audio

Здесь вы можете скачать аудио сигналы:

1. `Alg_1_0.wav` - синтезированное аудио
2. `real_0.wav` - настоящая речь

Далее у вас есть возможность подать данное аудио на вход функциям библиотеки

3.2.1 `predict_audio()`

Предположим, что ваш скачанный файл лежит на пути `file_path`. Тогда вы можете воспользоваться функцией `predict_audio()` следующим образом:

#Установка библиотеки

```
!pip install Pyara
```

#Импорт нужной нам функции

```
from pyara import predict_audio
```

#Подача аудио на вход функции

```
predict_audio('file_path/Alg_1_0.wav')
```

Функция вернет результат, показывающий, что аудиосигнал является синтезированным. Пример вывода показан на рисунке №2

```
predict_audio('C:/Users/79671/Downloads/Alg_1_0.wav')
```

```
Model Evaluated!  
Audio signal prepared !
```

```
1
```

Рисунок 2 – Вывод функции predict_audio при подаче синтезированной речи

Прделаем те же действия с аудиосигналом real_0.wav. Результат вы можете видеть на рисунке №3.

```
predict_audio('C:/Users/79671/Downloads/real_0.wav')
```

```
Model Evaluated!  
Audio signal prepared !
```

```
0
```

Рисунок 3 – Вывод функции predict_audio при подаче реальной речи

3.2.2 cut_if_necessary()

Если вы хотите самостоятельно обрезать аудио сигнал перед подачей его на вход модели, то вы можете воспользоваться функцией cut_if_necessary()

Пример использования данной функции показан ниже:

```
import torchaudio  
import torch  
MFCC_spectrogram = torchaudio.transforms.MFCC(  
    sample_rate=16000,  
    n_mfcc=80,  
    melkwargs={  
        "n_fft": 1024,  
        "n_mels": 80,  
        "hop_length": 256,  
        "mel_scale": "htk",  
        'win_length': 1024,
```

```

        'window_fn': torch.hann_window,
        'center': False
    },
)
from pyara import cut_if_necessary
signal = signal.mean(dim=0)
signal = signal.unsqueeze(dim=0)
signal, sample_rate = torchaudio.load('Alg_1_0.wav')
signal = MFCC_spectrogram(signal)
signal = cut_if_necessary(signal, 300)

```

На данном примере аудио сигнал real_0.wav загружается с помощью функции torchaudio.load, далее приводится к нормальному виду и преобразуется в MFCC спектрограмму. Данная спектрограмма подается на вход функции cut_if_necessary, где обрезается до размера 300 сэмплов. Результат работы функции вы можете видеть на рисунке 4.

```

import torchaudio
import torch
MFCC_spectrogram = torchaudio.transforms.MFCC(
    sample_rate=16000,
    n_mfcc=80,
    melkwargs={
        "n_fft": 1024,
        "n_mels": 80,
        "hop_length": 256,
        "mel_scale": "htk",
        "win_length": 1024,
        'window_fn': torch.hann_window,
        'center': False
    },
)
from pyara import cut_if_necessary
signal, sample_rate = torchaudio.load('C:/Users/79671/Downloads/real_0.wav')
signal = signal.mean(dim=0)
signal = signal.unsqueeze(dim=0)
signal = MFCC_spectrogram(signal)
signal = cut_if_necessary(signal, 300)
signal.shape

torch.Size([1, 80, 300])

```

Рисунок 4 – Результат работы функции cut_if_necessary

3.2.3 right_pad_if_necessary()

Если вы хотите самостоятельно увеличить размер аудио сигнала перед подачей его на вход модели, то вы можете воспользоваться функцией right_pad_if_necessary()

Пример использования данной функции показан ниже:

```

import torchaudio
import torch
MFCC_spectrogram = torchaudio.transforms.MFCC(
    sample_rate=16000,
    n_mfcc=80,
    melkwargs={
        "n_fft": 1024,
        "n_mels": 80,
        "hop_length": 256,
        "mel_scale": "htk",
        'win_length': 1024,
        'window_fn': torch.hann_window,
        'center': False
    },
)
from pyra import right_pad_if_necessary
signal, sample_rate = torchaudio.load('C:/Users/79671/Downloads/real_0.wav')
signal = signal.mean(dim=0)
signal = signal.unsqueeze(dim=0)
signal = MFCC_spectrogram(signal)
signal = right_pad_if_necessary(signal, 300)
signal.shape

```

На данном примере аудио сигнал `real_.wav` загружается с помощью функции `torchaudio.load`, далее приводится к нормальному виду и преобразуется в MFCC спектрограмму. Данная спектрограмма подается на вход функции `right_pad_if_necessary`, где расширяется как минимум до размера 300 сэмплов. Результат работы функции вы можете видеть на рисунке 4.

```

import torchaudio
import torch
MFCC_spectrogram = torchaudio.transforms.MFCC(
    sample_rate=16000,
    n_mfcc=80,
    melkwargs={
        "n_fft": 1024,
        "n_mels": 80,
        "hop_length": 256,
        "mel_scale": "htk",
        'win_length': 1024,
        'window_fn': torch.hann_window,
        'center': False
    },
)
from pyara import right_pad_if_necessary
signal, sample_rate = torchaudio.load('C:/Users/79671/Downloads/real_0.wav')
signal = signal.mean(dim=0)
signal = signal.unsqueeze(dim=0)
signal = MFCC_spectrogram(signal)
signal = right_pad_if_necessary(signal, 300)
signal.shape

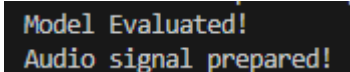
torch.Size([1, 80, 363])

```

Рисунок 5 – Результат работы функции right_pad_if_necessary

4 Сообщения

В процессе работы с библиотекой PyAra могут быть выданы следующие внутренние сообщения, представленные на рисунке 6.



```

Model Evaluated!
Audio signal prepared!

```

Рисунок 6 – Внутренние сообщения

Эти сообщения подтверждают правильность работы модели и отсутствия ошибок. Сообщение Model Evaluated показывает, что модель загружена верно и готова к обработке аудиосигнала. Сообщение Audio Signal prepared! показывает, что аудио сигнал успешно подготовлен специальными функциями. В противном случае библиотека выдаст ошибку.

Пример правильной работы функции для реального голоса представлен на Рисунке 7 – Пример обработки реального голоса


```
Model Evaluated!  
Audio signal prepared!  
Answer: 0 ; Probability: 1.0
```

Рисунок 7 – Пример обработки реального голоса

В данном примере ответ 0 означает, что это реальный голос с вероятностью 100% (данный пример из обучающего набора данных).

Пример работы функции для синтезированного голоса представлен на Рисунке 8 - Пример обработки синтезированного голоса

```
Model Evaluated!  
Audio signal prepared!  
Answer: 1 ; Probability: 0.96
```

Рисунок 8 – Пример обработки реального голоса

Сообщения об ошибках показываются пользователю в виде ошибок python, рассмотрим самые частые из них и обычные причины:

- Ошибка LibsndfileError – обычно возникает при неправильно указанном пути к файлу
- Ошибка LibsndfileError : Format not recognized – обычно возникает при неправильном формате входного аудиофайла. Библиотека работает с форматами mp3, wav, flac, aiff, ogg и протестирована на них. С остальными форматами могут возникать ошибки
- Ошибка FileNotFoundError – обычно возникает при неправильном пути к файлу с весами модели. Чтобы поставить модель с собственными весами необходимо заменить файл модели в функции `model_eval()` :
`weights_path = os.path.join(module_path, 'Model_weights.bin')`