Содержание

|  |  |
| --- | --- |
| [Аннотация](#page2) | 3 |
| [Ключевые слова](#page2) | 3 |
| [Введение](#page3) | 4 |
| [Обзор Литературы](#page4) | 5 |
| [Главы Проекта](#page6) | 7 |
| [Заключение](#page8) | 9 |
| [Список литературы](#page9) | 10 |
| [Приложения](#page9) | 11 |

2

Аннотация

* + создал программу для денойзинга аудио, которую можно запустить в терминале, и которая устраняет шумы из голосовых записей. В основе приложения лежит нейросетевая модель Demucs [[4](#page9)] (основанная на [[1](#page9)]), изначально реализованная на Python. Для повыше-ния производительности мы переписали модель на C++. В результате оптимизации кода нам удалось достичь значительного ускорения работы нейросети по сравнению demucs в пи-тоне. Был достигнут результат показателя RTF менее 0.25 и высокого SDR по сравнению
* чистым звуком — почти 10(протестировано на AMD Ryzen 7 8 ядер). Без многопоточно-сти RTF становится примерно 1.(В demucs на питоне RTF равен 2.5) Наше CLI-приложение позволяет пользователям загружать аудиофайлы в форматах WAV семпл рейта 16000 или 48000, очищать их от шумов и сохранять обработанный результат. Нам удалось разработать методику переноса моделей из Python в C++. Тестирование плагина на реальных записях подтвердило его эффективность — система успешно удаляет шумы.

Наше приложение может быть полезным для стримеров и подкастеров, стремящихся улучшить качество своих записей после эфира. Также этот код может быть интересен в чисто научных целях в изучении способ портирования кода модели с питона на си++.

Ключевые слова

денойзинг, VST-плагин, OBS Studio, JUCE, обработка,модель, Demucs, RTF

3

Введение

Основной целью данного проекта стала разработка CLI-приложения, использующего предобученные веса модели шумоподавления на основе архитектуры Demucs [[1](#page9)]. За осно-ву был взят существующий пример портирования нейросетевой модели Demucs с Python на C++[[7](#page9)], где авторы сохранили функциональность оригинальной реализации[[3](#page9)]. Обуче-ние модели уже было выполнено ранее, поэтому моя задача заключалась в адаптации её архитектуры для эффективной работы в аудиопотоке.

Для достижения цели потребовалось решить несколько ключевых задач.Необходимо было переписать класс модели на C++, чтобы обеспечить повышенную производительность при обработке звука в реальном времени. Это подразумевало корректную загрузку предо-бученных весов и точное воспроизведение последовательности слоёв, определённых в ориги-нальном Python-коде. Каждый слой (свёрточные операции, функции активации, нормализа-ция, LSTM) требовал сохранения реализации соответствующей версии в Torch в Python.

Также необходимо было распараллелить этот процесс используя потки си++. Клю-чевым являлось соображение, что можно сначало запустить какоето количество енкодеров параллельно, потом запустить лстмы последовательно(так как нужно передавать состояние hidden state и cell state между лстмами). И также потом запустить декодеры тоже параллель-но. Это можно делать последовательно для некоторого количества соседних чанков. Было показано, что чем больше размер одного чанка, тем выше SDR и RTF, но есть ограничение по памяти.(см 4 глава)

Для проверки корректности использовался метод поэтапного тестирования: я парал-лельно воспроизводил фрагменты кода на Python и C++, после чего сравнивал выходные тензоры в C++. Это позволило выявить и исправить расхождения в вычислениях, связан-ные с особенностями работы библиотеки Eigen в C++( [[8](#page9)]) и Torch в Python. Например, при считывании весов нужно было учитывать, что тензоры в Python хранятся в Fortran-style (по столбцам), а в C++ — в C-style (по строкам).

Был также проведен эксперимент в котором выяснилось, что для хаотичных зву-ков(наприер громкость шума менятся сильно в процессе или просто если разные звуки) sdr был отрицательным, что показывает, что для таких аудио это приложение будет плохим. Если же шум равномерный, то средний sdr по всем моим аудио получается больше 10, что и следовало ожидать от state of the art модели.(проверено на датасете с [[7](#page9)])

4

Обзор Литературы

Статья про Demucs(который я попытаюсь реализовать на плюсах) [[1](#page9)]:

Архитектура demucs состоит из L слоев энкодеров, потом идет LSTM и дальше L слоев де-кодеров. Каждый следующий энкодер увеличивает количество входов(нейронов) в два раза,

* то время как декодеры уменьшают наоборот в два раза. Также есть связи между попарно разбитыми на пары энкодерами и декодерами как на Рис. 0.1. H - количество выходных ка-налов из первого блока.

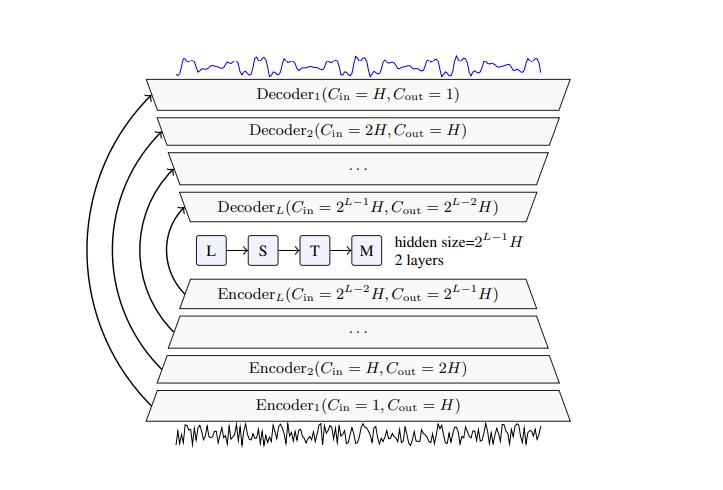


Рис. 0.1: Denoiser модель

Как видно из Рис. 0.2. ниже энкодеры состоят 2 одномерно сврточных сетей к которым сна-чало применили RELU, а к другому GLU. Параметры K и S это ядро свертки и шаг свертки соответственно. А декодер тоже самое, но наоборот. Также на схеме видно что часть выходов из энкодеров идет в декодеры.

Авторы также провели серию экспериментов по измерению RTF. Запустив модели на четы-рехъядерном Intel i5 CPU при H=64 полученный RTF=1.05, а при H=48 RTF=0.6. Во втором варианте если запускать только на одном ядре RTF=0.8.

5

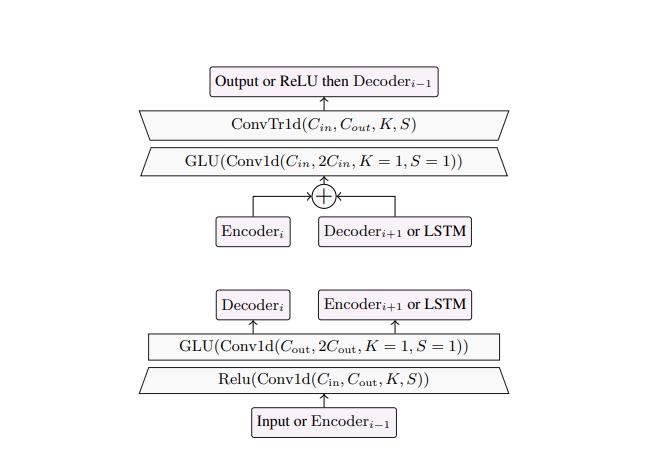


Рис. 0.2: Устройство изнутри для энкодеров и декодеров

Теперь перейдем к аналогам плагинов, которые денойзят аудио. Но всетаки нужно Таблица 0.1: Сравнение между собой разных плагинов для денойзинга по разным параметрам

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Uses ML | Real-Time | Paid/Free | Speech-Only | Open Source |
|  |  |  |  |  |  |
| Rx Voice Denoise | Yes | Yes | Paid | Yes | No |
| Waves Clarity VX | Yes | Yes | Paid | Yes | No |
| iZotope RX 10 | Yes | Yes | Paid | No | No |
| Bertom Denoiser | No | Yes | Free | Yes | Yes |
| ToneLib Noise Red. | No | Yes | Free | Yes | Yes |
| Моя реализация | Yes | No | Free | Yes | Yes |

заметить, что наш аналог перед ними отстает так оно не рилтайм, что усложняет взаимо-действие с этим плагином.

6

Главы проекта

Главы структурированы следующим образом: в Главе 1 Приводится причины выбора именно этих библитек и способа загрузки весов, в Главе 2 анализируется архитектура Demucs

* коде и тому как он строился, Глава 3 посвящена оптимизации вычислений на C++, в Главе

4 приведены результаты тестирования и результаты метрик.

* + I Глава Для начала обьясню как я пришел к выбору библиотек pytorchlib и Eigen. Eigen сам по себе является довольно популярной библиотекой для матричных вычис-лений. К тому же имеет полностью написанную с нуля библиотеке без использования сторонних библиотек, что упрощает ее подключение. Также в ней есть очень удобный тензорный класс который у меня лежит в tensors.h(как неймспейс). В некоторых биб-лиотеках, таких как Armadilio например, тензоров размера 4-5 нету,но в коде я часто создавал таковые. Минусом библиотеки Eigen является ограниченный функционал, хо-тя все необходимые функции для моей имплементации в ней есть(я бы сказал одной из самых полезных для сильного упрощения кода стала функция contract, которая произ-водит свертку тензора по какомуто другому тензору, а если быть точнее то прозводит суммирование по Эйнштейну, аналог einsum в numpy). Еще одним большим минусом является очень неподробная документация.(тут получше [[2](#page9)])

Способ загрузки весов было несколько. Изначально был план создать ggml ковертер, ко-торый веса в питоне бы сохранял в ggml файл [[5](#page9)] [[6](#page9)] и выгружал их в си++. Также был вариант использовать jit-trace, который сохранял полностью структуру класса модели

* + - все внутенние веса соответствующих модулей(слоев) из которых состояла модель и их можно было загрузить с помощью соответсвующей библиотеке jit из pytorchlib(си++). Я решил остановиться на втором варианте, так как второй вариант уже имел гото-вый конвертер torch.jit. Полученные веса можно было сгрузить в си++ используя одну из библиотек pytorchlib. Также мне показалось удобным, что можно итерироваться по слоям рекурсивно. Тоесть вся модель состояла из енкодера, декодера, лстма. А войдя рекурсивно из первой вершины в енкодер либо декодер можно было проитерироваться по слоям из которых они состояли. Я использовал такую структуру, чтобы сначало со-здать все мелкие компоненты модели, а потом из них уже сложить енкодер и декодер
    - всю модель(уже вместе с лстмом). Также так как jit модуль в си++ работает очень долго было решено создать собственный простой конвертер, который загружал бы веса каждых из слоев поочередно(также загружались размеры соответствующих тензоров)

7

* II Глава DemucsModel-это основная структура. Она состоит из полей: вектор из енко-деров(они инициализируются с разными размерами), вектор из декодеров(тоже), лстм, характеристики модели(такие как stride и kernel size например, которые уже потом передаются более мелким компонентам) и также например вектора skips, в котором пробрасываются выходы из енкодеров в декодеры(см. Рис. 0.1). Соответственно сна-чало вызываются енкодеры поочередно, потом лстм, который передает состояние cell state и hidden state следующему лстм) и потом декодеры. Енкодеры состоят из струк-тур conv1d, relu, glu. Декодеры из conv1d, glu, convtranspose1d. Соответственно загру-зить веса необходимо было только в conv1d, convtransposed1d, lstm(сами тензоры jit в основном имели вид weights-число или bias-число) Каждый из слоев conv1d, glu, convtranspose1d, relu, glu, lstm имеют собственные функции которые пробрасывают тензор, загружают с помощью jit веса, загружают веса(тензоры) в текстовый файл и функции считывающие с этих файлов веса и также функции которые проверяют, что сгруженные с текстого файла веса совпадают с теми которые изначально были. А для всех остальных структур(например OneEncoder) эти функции был реализованы итеративным вызовом сооответсвующих функций его компонент. Сама модель строи-лась последовательно. Проверялось, что каждый слой работает корректно с помощью кода на питоне, который был также организован в точности также и поэтому можно было сравнить результаты двух программ. Все тесты генерировались случайно и веса тоже, кроме последнего этапа, когда надо было с помощью функции dns48() в питоне загрузить веса для demucsmodel. Все тесты лежат в tests.cpp
* III Глава В певой имплементации DemucsModel дольше всех работали conv1d и contransposed1 Чтобы оптимизировать эти функции сначала была попытка использовать функцию convolve из eigen документации [[8](#page9)], что ускорило программу в 6 раз, что было недо-статочно. Потом была попытка оптимизировать с помощью флагов, таких как -Ofast или -ffast-math. Они ускорил программу еще в 2 раза(но только изначальную импле-ментацию, вторую она плохо оптимизировала). Последнее, что я сделал эта заново ре-ализовал функции пробрасывния у conv1d и contransposed1d, чтобы они использовали contract и все это вместе улучшило скорость программы гдето в 40 раз и сделал RTF по-чти 1. Последней оптимизацией было использование многопоточности, надо было лишь учесть, что необходимо передавать между соседними лстмами состояние hidden state, cell state(см. введение)
* IV Глава По итогу была проведена серия экспериментов(функция GenerateTestsforAudio

8

* test.cpp) и получен при количестве потоков 10 и размере чанка 16000 средний ре-зультат SDR больше 10(что считается хорошим результатом) и RTF примерно 0.25, что очень хорошо. В процессе еще было проверено, что результат почти не меняется при запускании одной и той же модели несколько раз. Такое возникло предположение, вследствие того , что состояние лстма(а именно cell state и hidden state) со временем кор-ректируется и возможно если сразу подавать его в модель,то она с самого начала лучше подавляет шум. Но наверное это ее "запоминание"происходит слишком быстро, чтобы это заметить. Также необходимо было ограничить размер обработанного чанка(так как чем больше чанк, тем лучше денойзит модель), так как иначе в лстме обрабатывались бы слишком большие для нее тензоры.(они бы не вместились в память). Поэтому было решено ограничится чанками размера около 16000(это примерно 1 секунда при семпл рейте 16000).

Заключение

Изначально планировалось, что также будет реализован OBS плагин. Тоесть конечная цель еще не реализована, но полученная готовая библиотека может быть использована в большом количестве других приложений. И она очень сильно оптимизирована, что ее вполне можно использовать в рил-тайме.

9

Список литературы

1. Yossi Ad Alexandre Defossez Gabriel Synnaeve. Real Time Speech Enhancement in the Waveform Domain. url: <https://arxiv.org/pdf/2006.12847>(дата обр. 06.09.2020).
2. Eigen Документация на гитлабе(more comprehensive). url: [https : / / gitlab . stce .](https://gitlab.stce.rwth-aachen.de/stce/eigen-ad/-/tree/43206ac4de09e348aaf0c8e7134f59872eaf55f4/unsupported/Eigen/CXX11/src/Tensor) [rwth-aachen.de/stce/eigen-ad/-/tree/43206ac4de09e348aaf0c8e7134f59872eaf55f4/](https://gitlab.stce.rwth-aachen.de/stce/eigen-ad/-/tree/43206ac4de09e348aaf0c8e7134f59872eaf55f4/unsupported/Eigen/CXX11/src/Tensor) [unsupported/Eigen/CXX11/src/Tensor](https://gitlab.stce.rwth-aachen.de/stce/eigen-ad/-/tree/43206ac4de09e348aaf0c8e7134f59872eaf55f4/unsupported/Eigen/CXX11/src/Tensor).
3. Facebook demucs. url: <https://github.com/facebookresearch/demucs>.
4. Facebook Denoiser. url: <https://github.com/facebookresearch/denoiser>.
5. Ggml документация в c(gguf формат). url: [https://github.com/ggml-org/ggml/blob/](https://github.com/ggml-org/ggml/blob/master/docs/gguf.md) [master/docs/gguf.md](https://github.com/ggml-org/ggml/blob/master/docs/gguf.md).
6. llama.cpp ggml чтобы импортировать с питона модели на си++. url: [https://github.](https://github.com/ggml-org/ggml/blob/master/docs/gguf.md) [com/ggml-org/ggml/blob/master/docs/gguf.md](https://github.com/ggml-org/ggml/blob/master/docs/gguf.md).
7. Имплемнтированный demucs.cpp. url: [https://github.com/sevagh/demucs.cpp/tree/](https://github.com/sevagh/demucs.cpp/tree/main) [main](https://github.com/sevagh/demucs.cpp/tree/main).
8. Официальная Eigen документация. url: [https://eigen.tuxfamily.org/dox/unsupported/](https://eigen.tuxfamily.org/dox/unsupported/eigen_tensors.html#title44) [eigen\_tensors.html#title44](https://eigen.tuxfamily.org/dox/unsupported/eigen_tensors.html#title44).

Приложения

RTF (Real-Time Factor): Параметр, показывающий скорость обработки аудио в реаль-ном времени. RTF менее 1.0 указывает на быструю обработку.

SDR (Signal-to-Distortion Ratio): Метрика качества аудиосигнала, измеряющая отно-шение полезного сигнала к искажениям. Высокий SDR(около 10) свидетельствует о хорошем качестве звука.

Eigen: C++ библиотека для линейной алгебры, используемая для работы с матрицами.

OBS (Open Broadcaster Software): Программное обеспечение с открытым исходным кодом для записи и потоковой передачи видео и аудио. OBS широко используется стриме-рами и подкастерами для создания и управления трансляциями, а также для интеграции различных источников медиа.

GGML (Gerganov’s General Machine Learning): C-библиотека для машинного обучения, особенно ориентированная на поддержку больших языковых моделей (LLMs) и высокопро-изводительных вычислений на обычном оборудовании. Она использует бинарный формат

10

для эффективного хранения и выполнения вычислений на центральном процессоре, что поз-воляет оптимизировать производительность и ресурсы при работе с большими моделями.

11