Санкт-Петербургский государственный университет

Выпускная квалификационная работа

общей образовательной программы бакалавриата
«Прикладная математика и информатика»

студента 4 курса, группы 18.Б04-мм
Ефима Сергеевича Бутакова

на тему

«Методы машинного обучения в задаче распознавания аудиосигнала»

Научный руководитель:
к.ф.-м.н., Михаил Сергеевич Ананьевский
кафедра теоретической кибернетики

Рецензент:
д.т.н., Игорь Борисович Фуртат
в.н.с. лаб. УСС ИПМаш РАН

г. Санкт-Петербург
2022

Saint Petersburg State University

Graduation Project

Applied Mathematics and Computer Science

Control and Processing of Information in Cybernetical and Robotic Systems

Butakov Efim

Machine Learning Techniques for Audio Recognition

Scientific Supervisor:

Mikhail Ananyevskiy

Candidate of Physico-Mathematical Sciences

Reviewer:

Igor Furtat

Doctor of Engineering Sciences

Saint Petersburg

2022

Оглавление

1. Введение

Автоматическое распознавание речи (Automatic speech recognition - ASR) – сфера, которая активно исследуется последние пять десятилетий, эта тема считается важным пунктом улучшения взаимодействия в сценариях человек - человек и человек-машина. Однако, в недалёком прошлом речь не являлась ключевым инструментом во взаимодействии человека и машины. Отчасти это связано с тем, что инструменты распознавания речи в то время были недостаточно хороши, чтобы преодолеть необходимый уровень точности и использоваться в реальных условиях. В то же время, альтернативные способы взаимодействия, например клавиатура и мышь, значительно превосходили речь по эффективности, ограничениям и точности.

В последние несколько лет наблюдается новый всплеск интереса к ASR. Это может быть связано с повышением требований к задаче автоматического распознавания речи в мобильных устройствах и успех новых виртуальных речевых помощников (например, Apple's Siri, Google Now и Microsoft's Cortana). Не менее важным пунктом является развитие методов глубокого обучения и увеличение вычислительных возможностей. Комбинированное использование методов глубокого обучения позволило уменьшить коэффициент ошибок в распознавании речи больше чем на треть относительно известных раннее классических методов GMM-HMM. Например, на данный момент точность распознавания слова для Английского языка достигает 95% и более.

Таким образом, объектом данного исследования является:задача ASR, предметом исследования является: использование актуальных архитектур нейронных сетей для задачи автоматического распознавания речи и сравнение их между собой.

1.1. Постановка задачи.

В рамках данной работы были поставлены следующие задачи:

- Исследовать современных подходы к решению задачи автоматического распознавания речи.
- Рассмотреть детали реализации этих моделей.
- Обучить несколько вариаций современных нейронных сетей.
- Сравнить обученные модели и выявить наиболее подходящий метод к решению задачи автоматического распознавания речи.

2. Обзор существующих методов.

2.1. Предобработка звука.

Подготовка звука.

Как известно звук — это волна. Но как превратить волну в набор чисел? Звуковые волны одномерны. В каждый момент времени у них есть одно значение, зависящее от амплитуды волны. Чтобы превратить звуковую волну в числа, мы просто записываем значения амплитуды волны в равноотстоящих точках. Этот процесс называется дискретизацией. Мы считываем данные тысячи раз в секунду и записываем числа, соответствующие амплитуде звуковой волны в этот момент времени. Чтобы не потерять

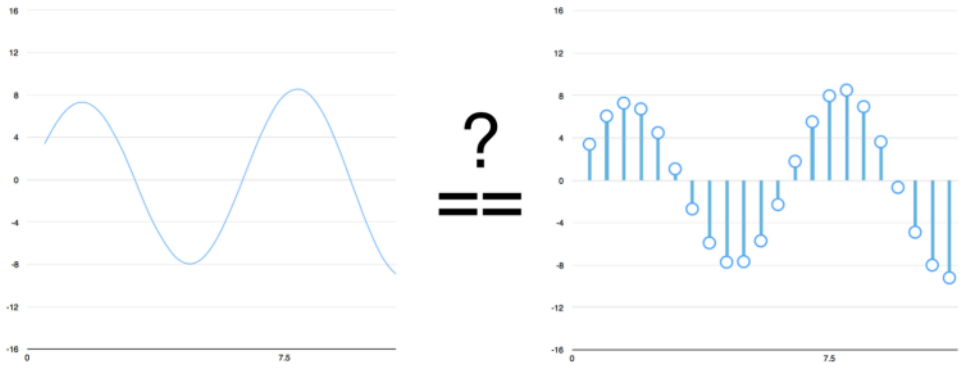

Рис. 1. Дискретизация.

часть информации мы воспользуемся теоремой Котельникова, которая говорит, что для идеального воссоздания исходной звуковой волны достаточно использовать частоту дискретизации, вдвое превышающую самую высокую частоту записываемого звука. Таким образом из звуковой волны мы получили большой массив чисел.

Создание спектограммы.

Мы могли бы просто обучить нейросеть на этих числах, но распознание речевых моделей путем обработки этих чисел напрямую затруднительно. Вместо этого мы можем облегчить задачу, проведя некоторую предварительную обработку аудиоданных. Для работы с цифровым звуком необходимо представить итоговую звуковую волну в виде суммы синусоидальных функций с помощью метода прямого преобразования Фурье, то есть найти спектр звукового сигнала.

Спектр звукового сигнала это совокупность синусоидальных звуковых волн, в результате суперпозиции образующих звуковую волну. Спектр звукового сигнала является самым важным инструментом анализа и обработки любого звука. Процесс преобразования звуковой волны в набор отдельных гармоник называется прямым преобразованием Фурье. Обратный процесс называется обратным преобразованием Фурье

Ряд Фурье для временной периодической функции $y(t)$:

$$y(t) = \frac{a_0}{2} + \sum_{i=1}^{\infty}(a_i cos(i) + b_i sin(i))$$

где

$$a_0 = \frac{2}{T} + \int_0^T y(t)dt$$

$$a_i = \frac{2}{T} + \int_0^T (y(t)cos(i\omega)dt$$

$$b_i = \frac{2}{T} + \int_0^T (y(t)sin(i\omega)dt$$

На примере произвольного фрагмента записи человеческой речи, записанного с помощью компьютера приведу пример спектра звукового сигнала и амплитудного графика. На следующем рисунке представлена некоторая зависимость амплитуды звукового сигнала от времени, которая характеризует определенный фрагмент человеческой представление ее в виде дв речи. Данный способ представления звука называется спек-

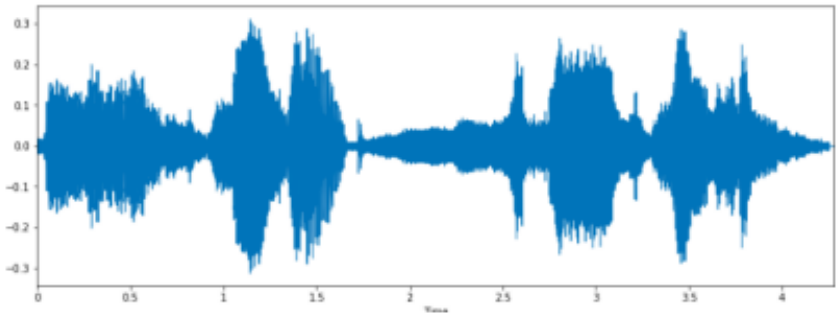

Рис. 2. Амплитудный график фрагмента человеческой речи..

трограммой звука или сигналограммой. Наиболее распространенным представлением спектрограммы является двумерной диаграммы:

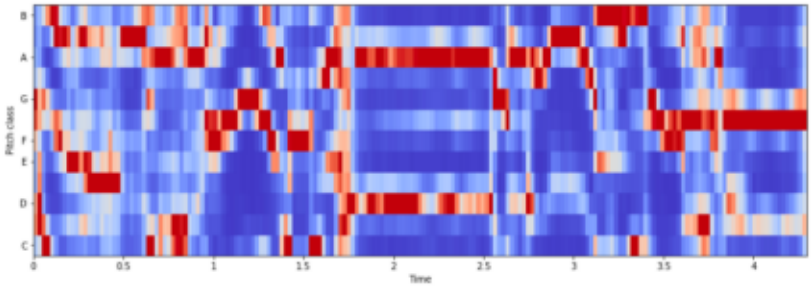

Рис. 3. Спектрограмма.

В данные работе для обучения и сравнения методов обучения нейронных сетей мы будем использовать спектограммы.

2.2. Сверточная нейронная сеть (CNN)

Сверточная нейронная сеть (англ. convolutional neural network, CNN) — специальная архитектура нейронных сетей, предложенная Яном Лекуном, изначально нацеленная на эффективное распознавание изображений. В сверточной нейронной сети выходы промежуточных слоев образуют матрицу (изображение) или набор матриц (несколько слоёв изображения). Так, например, на вход сверточной нейронной сети можно подавать три слоя изображения (R-, G-, B-каналы изображения). Основными видами слоев в сверточной нейронной сети являются сверточные слои (англ. convolutional layer), пулинговые слои (англ. pooling layer).

2.3. Рекуррентная нейронная сеть (RNN)

Начнем с основ, а именно с простого RNN.

Весь смысл обработки связной последовательности данных заключается в том, чтобы кроме выделения отклика для каждого элемента, суметь учесть и связь элементов. Например, можно представить изображение в виде набора векторов, каждый из которых содержит пиксели одной колонки. Будем последовательно подавать такие вектора на вход сети, если мы хотим, чтобы сеть классифицировала изображение, необходимо сделать так, чтобы сеть умела “запоминать” информацию об уже просмотренных колонках. Аналогично, если мы хотим научить сеть классифицировать предложения естественного языка (например, определять эмоциональный окрас предложения), и подаём в сеть одно слово за другим, нам желательно, чтобы сеть “помнила” уже переданные

слова. Если мы хотим, чтобы сеть переводила предложение с одного языка на другой, то тоже было бы не плохо учитывать начало предложение при переводе середины и конца.

Как раз задачу “запоминания” уже отсмотренных элементов последовательности и предполагается решать при помощи рекуррентной сети. Для этого, кроме выходного вектора, сеть должна иметь еще и некоторый вектор, который описывает текущее внутреннее состояние сети, т.е. в нем содержатся воспоминания о всех уже отсмотренных сетью элементах. Более формально это выглядит так.

Допустим у нас есть набор входных векторов $(X^{(1)}, X^{(2)}, \ldots, X^{(n)})$, мы их последовательно преобразуем:

$$H^t = \sigma(W^{hx}X^t + W^{hh}H^{(t-1)} + b_h)$$

$$Y^{(t)} = W^{yh}H^{(t)} + b_y$$

При этом кроме выхода $Y(t)$ (который нам возможно и не нужен на каждом шаге), мы имеем еще вектор $H(t)$, описывающий текущее состояние.

Таким образом сеть выглядит следующим образом:

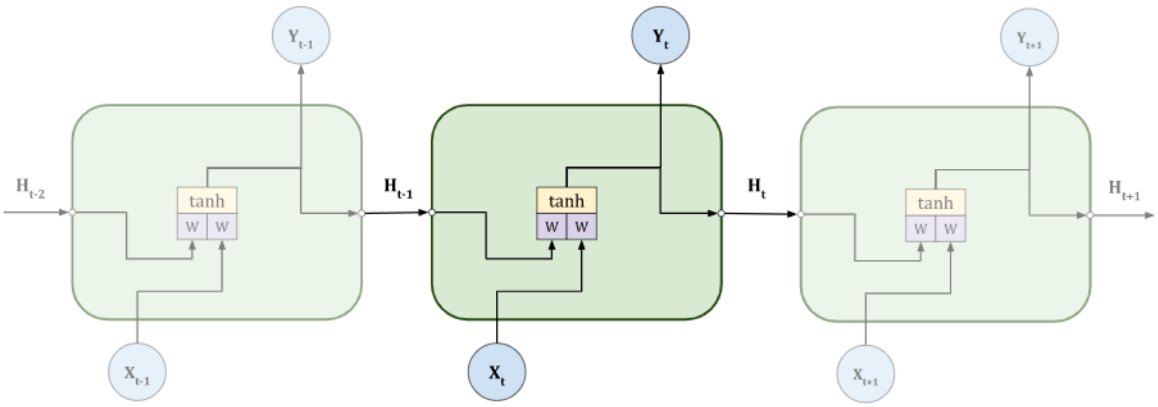

Рис. 3. RNN.

2.4. Сети с долгой кратковременной памятью (LSTM)

Проблема с обычными RNN ячейками, описанными выше, заключается в том, что они не могут “удержать в памяти” слишком длинные последовательности. Связано это

с тем, что когда мы прокидываем градиенты через, достаточно, длинную последовательность, то мы сталкиваемся с одной из двух проблем: либо градиенты уменьшаются настолько, что ошибки на конце последовательности перестают влиять на ее начало, либо градиенты увеличиваются, и процесс расходится.

Чтобы победить эту проблему было предложено заменить обычные RNN ячейки на более продвинутый вариант LSTM. В базовом варианте LSTM было добавлено еще одно внутренние состояние ячейки S_t, которое на каждом шаге преобразовывается, и за счет этого гасится проблема исчезающих или взрывающихся градиентов.

Так же в LSTM ячейку были добавлены input и output gate, которые должны были воздействовать на то, что из входных данных окажет влияние на внутренние состояние ячейки и что из внутреннего состояния должно отправится на выход. Также после этого начили добавлять forget gate, которое позволяет занулять некоторые компоненты внутреннего состояния $S^{(t-1)}$ прежде чем их передовать дальше. В итоге получаем следующую ячейку LSTM сети:

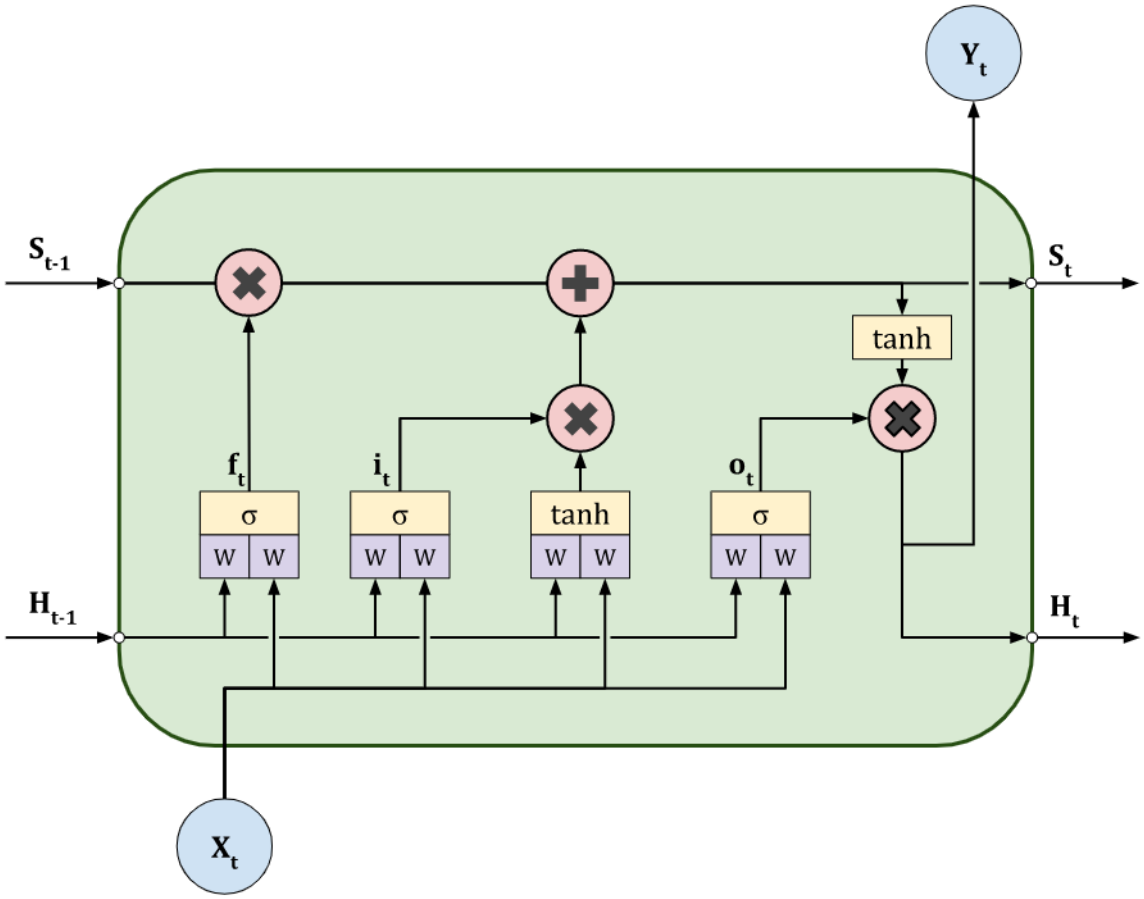

Рис. 4. Ячейка LSTM сети.

Формально обучение происходит следующим образом:

$$g^{(t)} = \phi(W^{gx}X^{(t)} + W^{gh}H^{(t-1)+b^g})$$

$$i^{(t)} = \sigma(W^{ix}X^{(t)} + W^{ih}H^{(t-1)+b^i})$$

$$o^{(t)} = \sigma(W^{ox}X^{(t)} + W^{ih}H^{(t-1)+b^o})$$

$$f^{(t)} = \sigma(W^{fx}X^{(t)} + W^{fh}H^{(t-1)+b^f})$$

$$S^{(t)} = g^{(t)} * i^t + S^{(t-1)} * f^t$$

$$H^{(t)} = \phi(S^{(t)}) * o^{(t)}$$

где $i^{(t)}, o^t$ это соответсвенно input и output gate,

2.5. Закрытый рекуррентный блок GRU

Еще один вариант ячейки, очень похожий на LSTM, но с меньшим количеством gate (двумя), а значит и параметров (качество при этом обещают сравнимое с LSTM с таким же количеством параметров):

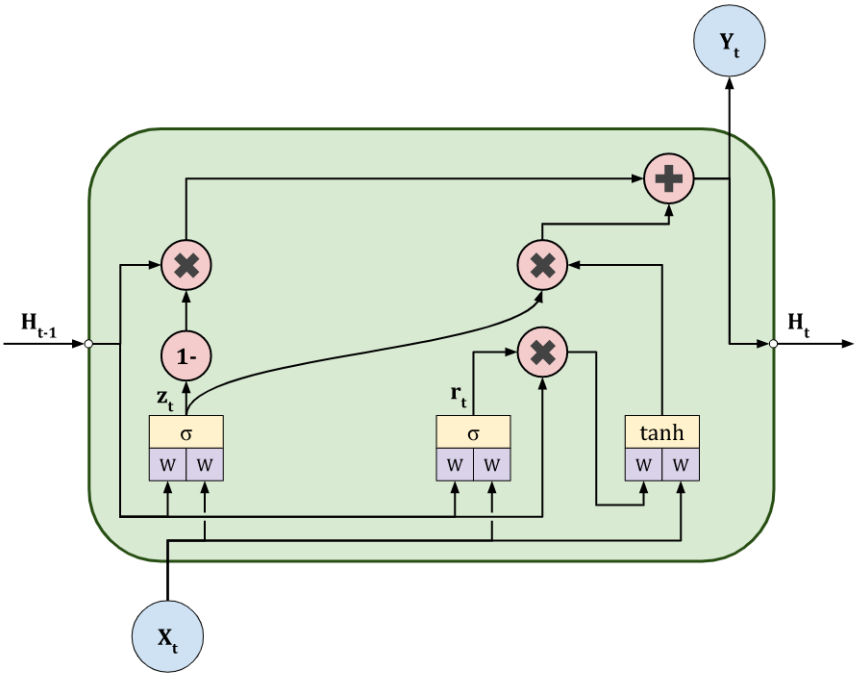

Рис. 5. Ячейка GRU.

Формально обучение происходит следующим образом:

$$z^{(t)} = \sigma(W^{zx}X^{(t)} + W^{zh}H^{(t-1)+b^z})$$

$$r^{(t)} = \sigma(W^{rx}X^{(t)} + W^{rh}H^{(t-1)+b^r})$$

$$\widetilde{H}^{(t)} = \phi(W^{\widetilde{h}x}X^t + W^{\widetilde{h}h}(r^t * H^{(t-1)}) + b_{\widetilde{w}})$$

$$H^{(t)} = (1 - z^{(t)}) * H^{(t)-1} + z^{(t)} * \widetilde{h}^{(t)}$$

2.6. DeepSpeech

Следующим этапом в рассмотрении известных подходов к решению задачи автоматического распознавания речи является обзор наиболее знаковых систем и архитектур, с помощью которых производится наиболее эффективное распознавание речи в последние годы. Без сомнения к таким архитектурам могут быть отнесены DeepSpeech, DeepSpeech2 и QuartzNet.

DeepSpeech представляет собой сквозную (end-to-end) систему для автоматического распознавания речи, которая является более простой относительно классических подходов (GMM и HMM) и достигает лучших результатов. Основа архитектурыпредставляет собой рекуррентную нейронную сеть (RNN), которая обучается напрямую из данных и не требует настройки отдельных компонент для адаптации спикера или фильтрации шума. DeepSpeech (2014) опережает по точности раннее опубликованные методы на корпусе $SwitchboardHub5'00$ и достигает 16.0% ошибок. Рекуррентная нейронная сеть (RNN) обучена на восприятие спектрограмм и её целью является генерация текстовых транскрипций ко входящим аудио. Пусть одна запись x и её метка (транскрипция) y взяты из тренировочного набора $X = (x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ...$ Каждая запись $x^{(i)}$ - это временной ряд длины $T^{(i)}$, где каждый временной срез - это вектор аудио признаков, $x_t^{(i)}$, $t = 1, ...T^{(i)}$. В качестве аудиопризнаков используются спектрограммы, таким образом $x_{(t,p)}^{(i)}$ обозначает мощность p-ого элемента по частоте в момент времени t в аудио записи. Целью RNN является преобразование входной последовательности x в вероятностную последовательность символов для транскрипции y,

$$\hat{y}_t = P(c_t|x), c_t \in a, b, c, ...z, space, apostrophe, blank$$

Модель RNN состоит из 5 скрытых состояний. Для входа x скрытые состояния на слое l обозначаются как $h^{(l)}$, то есть что $h^{(0)}$ это входное состояние. Первые три

слоя RNN не являются рекуррентными. Для первого слоя, в каждый момент времени t выходные данные зависят от кадра спектрограммы x_t вместе с контекстом из $C \in 5, 7, 9$ кадров. Второй и третий слои работают с независимыми данными для каждого шага (без использования контекста).

Таким образом, для каждого момента времени t первые три слоя определяются следующим образом:

$$h_t^{(l)} = g(W^{(l)} h_t^{(l-1)} + b^{(l)})$$

где $g(z) = min(max(0, z), 20)$ - это ограниченная сверху функция активации ReLu (rectified-linear unit) и $W^{(l)}, b^l$ - матрица весов и параметр смещения для слоя l. Четвертый слой – это двунаправленный рекуррентный слой. Этот слой содержит два подмножества скрытых состояний: с прямой рекуррентностью $h^{(f)}$ и с обратной $h^{(b)}$:

$$h_t^{(f)} = g(W^{(4)} h_t^{(3)} + W_r^{(f)} h_(^{(f)} t - 1) + b^{(4)})$$

$$h_t^{(b)} = g(W^{(4)} h_t^{(3)} + W_r^{(b)} h_(^{(b)} t + 1) + b^{(4)})$$

Стоит заметить, что h^f должен быть вычислен последовательно с $t = 1$ до $t = T^{(i)}$ для i-ой записи, в то время как $h^{(b)}$ должен быть вычислен последовательно в инверсированном порядке с $t = 1$ до $t = T^{(i)}$. Пятый (не рекуррентный) слой принимает на вход скрытые состояния с прямой и обратной рекуррентностью $h_t^{(4)} = g(W^{(5)} h_t^{(4)} + b^{(5)})$, где $h_t^{(4)} = h_t^{(f)} + h_t^{(b)}$. Результатом данного слоя является стандартная функция softmax, которая вычисляет вероятности предсказанных символов для каждого временного интервала t и символа k из алфавита:

$$h_{t,k}^{(4)} = \hat{y_{t,k}} \equiv P(c_t = k|x) = \frac{exp(W^{(6)} h_t^{(5)} + b^{(6)}}{\sum W_j^{(5)} h_t^{(5)} + b_j^{(6)}}$$

где $W_k^{(6)}$ и $b_k^{(6)}$ обозначают k-ый столбец матрицы весов и k-е смещение (bias), соответственно. Полная модель RNN, использованной в архитектуре DeepSpeech проиллюстрирована на рис. 6.

2.7. DeepSpeech 2

Следующей ступенью эволюции DeepSpeech стала архитектура DeepSpeech 2 , которая в своей основе так же имела RNN, кроме того, авторами была рассмотрена

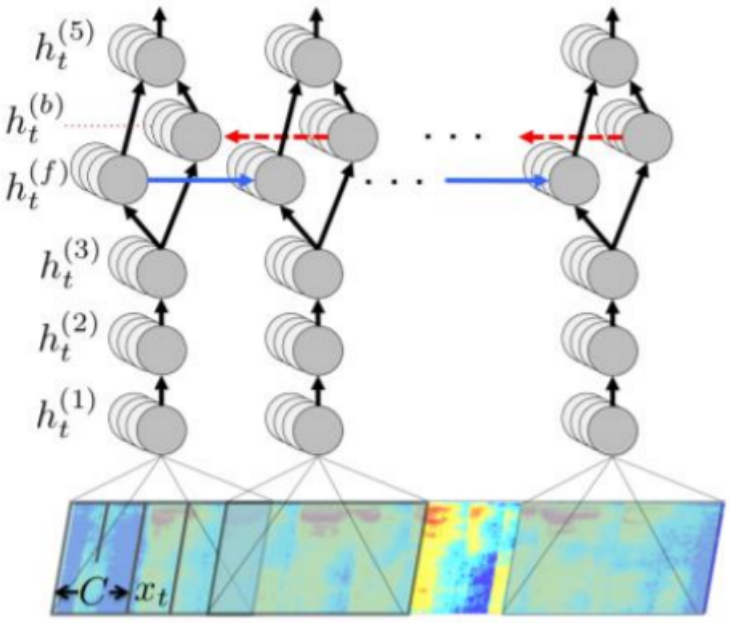

Рис. 6. Архитектура DeepSpeech.

возможность адаптации модели под несколько языков (Английский, Китайский). На рис. 7 представлена архитектура DeepSpeech 2, нетрудно заметить отличия от архитектуры её предшественника – DeepSpeech: добавлены нормализация по батчам (Batch Normalization), инвариантные свёртки к входным спектрограммам и один полносвязный слой перед применением softmaxслоя. Данная модель по прежнему тренируется с помощью CTC функции потерь , которая позволяет напрямую предсказывать последовательность символов из входного аудио. Кроме того, в качестве основного блока сети были рассмотрены GRU (Gated Recurrent Units) и LSTM (Long Short-Term Memory).

В качестве дальнейшего исследования авторами была выбрана модель GRU, так как в экспериментах с малыми наборами данных GRU и LSTM достигают одинаковой точности со схожим количеством параметров, при этом GRU имеет более высокую скорость обучения и меньшую вероятность расхождения. Также авторы обращают внимание на использование пакетной нормализации (batch normalization) при обучении глубоких рекуррентных нейронных сетей (Deep RNNs). При увеличении числа рекуррентных слоёв в RNN увеличивается время тренировки сети, так как глубина и размер модели увеличивается. Авторы DeepSpeech 2 также провели эксперименты для исследования возможности ускорения тренировки основного блока системы. Результаты тестирования DeepSpeech2 демонстрируют, что система хорошо справляется с аудио, которые представляют собой прочитанный вслух текст (строка Read) и для $\frac{3}{4}$ наборов

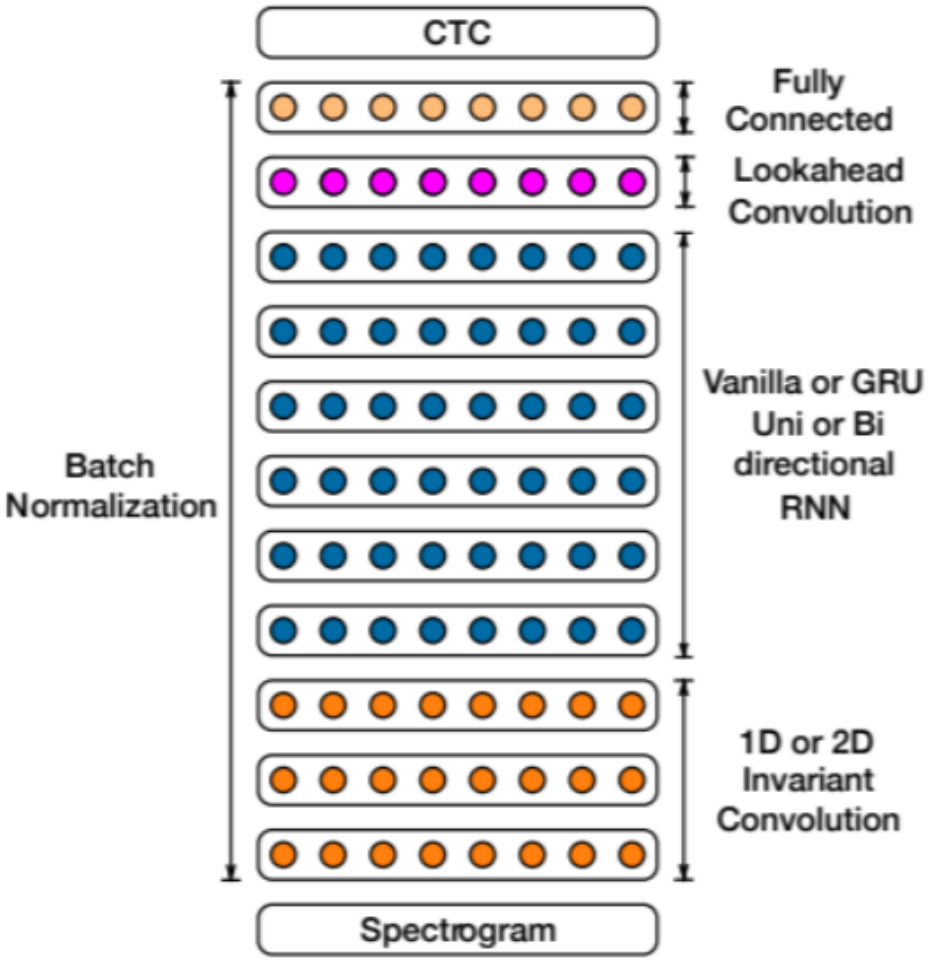

Рис. 7. Архитектура DeepSpeech 2.

данных опережает по точности человека. В то же время, на шумных данных (строка Noisy) модель показывает существенное отставание от результатов показанных человеком, в среднем, на 10% - это говорит о том, что исследование темы автоматического распознавания на шумных данных актуально и на данный момент, так как DeepSpeech 2 по прежнему активно используется для коммерческого распознавания речи и имеет множество адаптаций под разные языки.

2.8. QuartzNet

Следующая архитектура, которая будет рассмотрена имеет название QuartzNet была представлена в 2019 году. QuartzNet представляет собой сквозную (end-to-end) нейро-акустическую модель для автоматического распознавания речи, эта архитектура представляет собой множество блоков, которые соединены остаточными связями (residual connections). Каждый блок состоит из одного или нескольких модулей с одномерными свёртками, разделёнными по времени (1D time-channel separable convolutional layer), пакетной нормализацией и ReLU (rectified-linear unit) слоями. Модель обучена с помощью CTC (Connectionist Temporal Classification) функции ошибок.Воснову архитектуры QuartzNet легла свёрточная модель JasperNet (2019). Основным нововведением архитектуры QuartzNet является замена обычных одномерных свёрток на одномерные свёртки разделяемые по времени с реализацией разделения по глубине (depthwise separable convolutions). Польза этой модификации объясняется тем, что одномерные свертки с разделением на временные каналы могут быть разделены на одномерный сверточный слой по глубине с длиной ядра K который работает на каждом канале индивидуально, но через K временных кадров, и точечный сверточный слой, который работает на каждом временном кадре независимо, но по всем каналам.

Модель QuartzNet имеет следующую структуру: свёрточный одномерный слой C_1, за которым следует последовательность блоков B.

Каждый блок B_i повторяется S_i и имеет остаточные связи (residual connections) между блоками. Каждый блок B_i состоит из одинаковых базовых модулей, которые повторяются R_i раз и включают в себя четыре слоя:

- Глубинный свёрточный слой размера K с каналами c_{out}
- Поточечная свёртка

- Слой нормализации
- ReLU

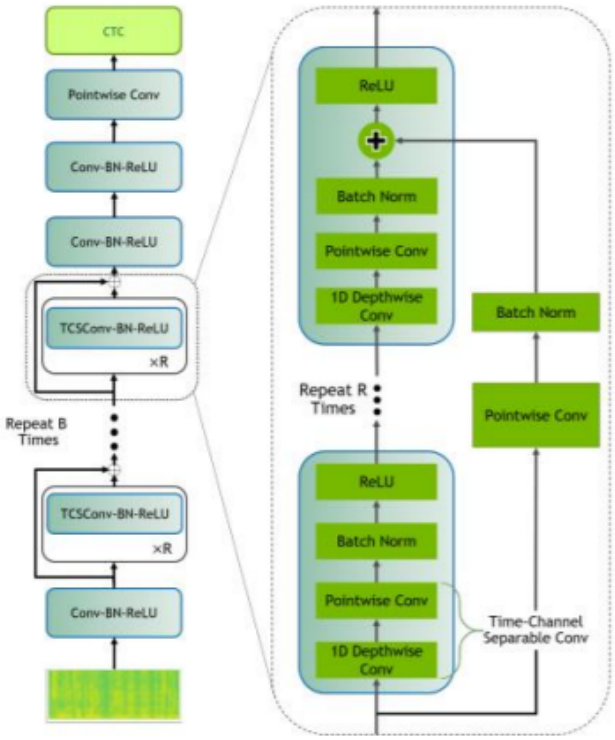

Рис. 8. Архитектура модели QuartzNet

Для нейросетевой акустической модели одним из определяющих параметров является количество параметров, эта характеристика влияет на скорость обучения, время распознавания и адаптивность модели к аппаратному обеспечению с различными ограничениями. Авторы модели QuartzNet предложили модифицировать один из блоков, пронаблюдать за метрикой точности и выяснить: как уменьшение числа параметров влияет на точность акустической модели.

Одномерный свёрточный (1D convolutional layer) слой имеет количество параметров равное произведению K $\times$ c_{in} $\times$ c_{out}, где K – размер ядра свёртки, c_{in} - число входных каналов и c_{out} – число выходных каналов. Разделяемые по временным каналам свёртки имеют количество параметров равное K $\times$ c_{in} + c_{in} $\times$ c_{out}. Поскольку K в большинстве случаев в несколько раз меньше, чем c_{out} рассматривается возможность использования групповых свёрток для этого модуля и последующий обмен информацией между группами свёрток (рис. 9).

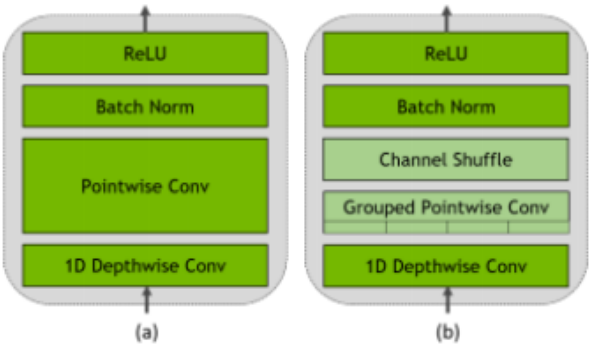

Рис. 9. (a) Разделяемые по временным каналам свертки (b) Разделяемые по временным каналам свертки с разделением по группам и дальнейшим обменом информации

Применение данного метода модификации для свёрточного модуля позволило уменьшить количество параметров модели более чем в два раза (с 18.9 до 8.7), при этом ухудшение метрики WER для максимального числа групп составило менее 1%. Лучшие

# Groups	dev-clean	dev-other	Params, M
1	3.98	11.58	18.9
2	4.29	12.52	12.1
4	4.51	13.48	8.70

Рис. 10. Результаты тестирования (WER метрика) модели QuartzNet-15×5 с применением групповых свёрток (число групп: 1, 2, 4), для обучения использовался набор данных LibriSpeech, 300 эпох обучения. Тестирование на подвыборках dev (clean and other)

результаты на наборе данных LibriSpeech достигнуты с моделью QuartzNet-15x5, которая состоит из пятнадцати блоков с пятью свёрточными модулями на каждый блок. Модель с разделяемыми по временным каналам свёртками, описанная выше, имеет гораздо меньшее число параметров, чем модель с обыкновенными свёртками и менее склонна к чрезмерному переобучению, поэтому в качестве регуляризации авторы используют только расширение набора данных (data augmentation) и сокращения веса (weight decay).

3. Эксперименты по обучению моделей.

3.1. Данные для распознавания речи LJ Speech Dataset.

LJ Speech Dataset - это общедоступный набор речевых данных, состоящий из 13 100 коротких аудиоклипов, в которых один спикер читает отрывки из 7 научно-популярных книг. Транскрипция предоставляется для каждого клипа. Клипы различаются по длине от 1 до 10 секунд и имеют общую продолжительность примерно 24 часа.

Тексты были опубликованы между 1884 и 1964 годами. Аудио было записано в 2016-17 годах проектом LibriVox и также находится в открытом доступе.

Статистика:

- Всего клипов - 13,100
- Всего слов - 225,715
- Всего персонажей - 1,308,678
- Общая продолжительность - 23:55:17
- Средняя продолжительность клипа - 1,11 сек.
- Отдельные слова - 13,821

3.2. Функция ошибки.

В системах автоматического распознавания речи основным показателем качества является точность распознавания, которая определяется как процент правильно распознанных слов (WRR — Word Recognition Rate) или, н-оборот, неправильно распознанных слов (WER — Word Error Rate). Иногда также используется показатель ошибок распознавания фраз/предложений (SER — Sentence Error Rate), который является важным в диалоговых системах, где корректировка гипотезы распознавания невозможна в отличие от задачи диктовки текста. В последнее время в качестве основного показателя точности работы систем распознавания речи используется показатель WER, а именно, его абсолютное значение или относительное, если сравниваются различные модели/системы. Поскольку с развитием речевых технологий показатель WER все более приближается к нулю, то улучшение его значения более наглядно, чем повышение точности распознавания слов. Метод определения показателя WER состоит в выравнивании двух

текстовых строк (первая — это результат распознавания, а вторая — запись того, что было сказано в действительности) с помощью алгоритма динамического программирования с вычислением расстояния Левенштейна. Расстояние Левенштейна представляет собой „стоимость“ редактирования данных (минимальное количество или взвешенная сумма операций редактирования) для преобразования первой строки во вторую с наименьшим числом операций ручной замены (S), удаления (D) и вставки (I) слов:

$$WER = \frac{S + D + I}{T},\ WRR = 1 - WER$$

где T — количество слов в распознаваемой фразе. В данной работе мы будет сравнивать моделе с помощью WER.

3.3. Тестирование и сравнение моделей.

В качестве первой модели была выбрана модель с двумя сверточными слоями и одним полносвязным слоем. После 50ти эпох обучений с помощью CTC (Connectionist Temporal Classification) функции ошибок модель показала $WER = 0.6$ Что является очень плохим результатом.

Следующим этапом были выбраны модели на основе DeepSpeech:

- 2 сверточных слоя нейронной сети, 2 слоя GRU, обучаемая с помощью CTC функции ошибок и Dropout 50%. Модель уже имеет порядка 5.6 миллиона параметров и после 50ой эпохи имела $WER = 0.37$
- 2 сверточных слоя нейронной сети, 2 слоя $LSTM$, обучаемая с помощью CTC функции ошибок и Dropout 50%. Здесь в качестве эксперимента 2слоя GRU были заменены на слои $LSTM$. Что в теории при большем количестве параметров и более длительном обучении, модель должна была показать результат лечше, чем у предыдущей модели. Модель уже имеет порядка 7.6 миллиона параметров и после 50ой эпохи имела $WER = 0.29$

Позже были обучены современные методы автоматического распознавания речи на основе DeepSpeech 2 и QuartzNet. А именно:

- DeepSpeech 2 с 5ю слоями GRU внутри, обучаемая с помощью CTC функции ошибок, Dropout 50%, а также наложенным шумом. Модель имеет порядка 26 миллионов параметров и после 50ой эпохи имела $WER = 0.17$

- DeepSpeech 2 с 5ю слоями LSTM внутри, обучаемая с помощью CTC функции ошибок, Dropout 50%, а также наложенным шумом. Здесь в качестве эксперимента 5 слов GRU были заменены на слои LSTM. Модель уже имеет порядка 35 миллиона параметров и после 50ой эпохи имела $WER = 0.13$
- Нейросеть jasper_10x5 с 330 миллионами параметров и $WER = 0.087$
- Нейросеть quartznet15x5 с 19 миллионами параметров и $WER = 0.074$

Модель	Количество параметров	WER
2 Слоя CNN + Dense + CTC Loss + Drop(0.5)	1.9M	0.6
DeepSpeech: 2 Слоя CNN + 2 слоя GRU + CTC Loss + Drop(0.5)	5.6M	0.37
DeepSpeech: 2 Слоя CNN + 2 слоя LSTM + CTC Loss + Drop(0.5)	7.6M	0.29
DeepSpeech 2: 2 Слоя CNN + 5 слоев LSTM + CTC Loss + Drop(0.5) + Noise + shifter + stretch	35M	0.13
DeepSpeech 2: 2 Слоя CNN + 5 слоев GRU + CTC Loss + Drop(0.5) + Noise + shifter + stretch	26M	0.17
quartznet15x5	19M	0.074
jasper_10x5	330M	0.087

Рис. 11. Результаты тестирования моделей.

Результаты тестирования моделей представленные на рис. 11 демонстрируют способность модели QuartzNet достигать современных и релевантных результатов в области автоматического распознавания речи с использованием модели с меньшим числом параметров.

3.4. Возможные улучшения полученных результатов

В качестве улучшения могут быть рассмотрены несколько вариантов: увеличение количества обучающих данных, увеличение вычислительных мощностей, использование нейросетевыхмоделей в качестве лингвистической модели, например Transformer-XL.

4. Результаты.

Проведённый обзор современных подходов позволяет оценить подходы к решению задачи автоматического распознавания речи с помощью использования нейросетевых акустических моделей. Анализ архитектур рассмотренных моделей (GRU, LSTM, DeepSpeech, DeepSpeech 2 и QuartzNet) и результаты, полученные при их тестировании, позволяют сделать вывод: модель QuartzNet является наиболее подходящей архитектурой для решения задачи автоматического распознавания речи ввиду меньшего числа параметров и при этом высоких показателей при распознавании (WER).

В ходе выполнения работы были исследованы современные подходы к решению задачи автоматического распознавания речи, рассмотрены детали реализации этих моделей, обучены несколько вариаций нейронных сетей с последующим сравнением между собой.

Полученные модели могут быть использованы для распознавания речи в реальной среде и доработаны для решения реальных задач в сфере автоматического распознавания речи.

5. Список литературы.

1. Levenshtein V. I. Binary codes capable of correcting deletions, insertions and reversals // Sov. Phys. Dokl. 1966. Vol. 6. P. 707—710.

2. Wang, Y.; Acero, A.; Chelba, C. (2003). Is Word Error Rate a Good Indicator for Spoken Language Understanding Accuracy. IEEE Workshop on Automatic Speech Recognition and Understanding. St. Thomas, US Virgin Islands. CiteSeerX 10.1.1.89.424.

3. ImageNet Classification with Deep Convolutional Neural Networks / Alex Krizhevsky [et al.]. –November 2013, 9 S.

4. Juang, B. H., Rabiner, L. R. (1991). Hidden Markov models for speech recognition. Technometrics, 33(3), 251-272.

5. Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., ... Ng, A. Y. (2014). Deep speech: Scaling up end-to-end speech recognition. arXiv preprint arXiv:1412.5567.

6. Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., ... Zhu, Z. (2016, June). Deep speech 2: End-to-end speech recognition in english and mandarin. In International conference on machine learning (pp. 173-182). PMLR.

7. Kriman, S., Beliaev, S., Ginsburg, B., Huang, J., Kuchaiev, O., Lavrukhin, V., ... Zhang, Y. (2020, May). Quartznet: Deep automatic speech recognition with 1d time-channel separable convolutions. In ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 6124-6128). IEEE.

8. M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11):2673–2681, 1997

9. Graves, A., Fernández, S., Gomez, F., and Schmidhuber, J. Connectionist 36 temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In ICML, pp. 369–376. ACM, 2006.

10. Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. Neural Computation, 9(8):1735—1780, 1997.

11. A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, ´ "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in ICML, 2006.

12. J. Li, V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J.M. Cohen, H. Nguyen, and R.T. Gadde, "Jasper: An end-to-end convolutional neural acoustic model," arXiv:1904.03288, 2019

13. Kuchaiev, O., Li, J., Nguyen, H., Hrinchuk, O., Leary, R., Ginsburg, B., ... Cohen, J. M. (2019). Nemo: a toolkit for building ai applications using neural modules. arXiv preprint arXiv:1909.09577.

14. NVIDIA, QuartzNet15x5Base-En, - https://ngc.nvidia.com/catalog/models/

15. Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

16. Bracewell, R. N., Bracewell, R. N. (1986). The Fourier transform and its applications(Vol. 31999, pp. 267-272). New York: McGraw-Hill.

6. Приложение

```
gpu_info = !nvidia-smi
gpu_info = '\n'.join(gpu_info)
if gpu_info.find('failed') >= 0:
  print('Not connected to a GPU')
else:
  print(gpu_info)

from psutil import virtual_memory
ram_gb = virtual_memory().total / 1e9
print('Your runtime has {:.1f} gigabytes of available RAM\n'.format(ram_gb))

if ram_gb < 20:
  print('Not using a high-RAM runtime')
else:
  print('You are using a high-RAM runtime!')

!pip install jiwer

import pandas as pd
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import matplotlib.pyplot as plt
from IPython import display
from jiwer import wer
from tensorflow.keras import backend as K
from tqdm.notebook import tqdm

data_url = "https://data.keithito.com/data/speech/LJSpeech-1.1.tar.bz2"
data_path = keras.utils.get_file("LJSpeech-1.1", data_url, untar=True)
wavs_path = data_path + "/wavs/"
metadata_path = data_path + "/metadata.csv"


# Read metadata file and parse it
metadata_df = pd.read_csv(metadata_path, sep="|", header=None, quoting=3)
metadata_df.columns = ["file_name", "transcription", "normalized_transcription"]
metadata_df = metadata_df[["file_name", "normalized_transcription"]]
metadata_df = metadata_df.sample(frac=1, random_state = 228).reset_index(drop=True)
metadata_df.head(3)

split = int(len(metadata_df) * 0.85)
df_train = metadata_df[:split]
df_val = metadata_df[split:]
```

```
46
47  print(f"Size of the training set: {len(df_train)}")
48  print(f"Size of the training set: {len(df_val)}")
49
50  for x in tqdm(df_val['file_name']):
51      !cp /root/.keras/datasets/LJSpeech-1.1/wavs/{x}.wav /content/OpenSeq2Seq/{x}.wav
52
53  wer()
54
55  test = pd.DataFrame()
56  test['wav_filename'] = df_val['file_name'].tolist()
57  test['wav_filename'] = test['wav_filename'] + '.wav'
58  test['wav_filesize'] = 'UNUSED'
59  test['transcript'] = 'UNUSED'
60  test.to_csv('/content/OpenSeq2Seq/test.csv', index = None)
61
62  test
63
64  %tensorflow_version 1.x
65
66  import os
67  from os.path import exists, join, basename, splitext
68
69  git_repo_url = 'https://github.com/NVIDIA/OpenSeq2Seq.git'
70  project_name = splitext(basename(git_repo_url))[0]
71  if not exists(project_name):
72    # clone and install dependencies
73    !git clone -q --depth 1 {git_repo_url}
74    !git checkout e958b7d
75    !pip uninstall -y -q pymc3
76    !pip install --upgrade joblib
77    #!cd {project_name} && pip install -q -r requirements.txt
78    !pip install -q youtube-dl librosa python_speech_features sentencepiece
79    !pip install -q --upgrade gdown
80
81    # create eval config
82    !cp {project_name}/example_configs/speech2text/jasper10x5_LibriSpeech_nvgrad.py {project_name}/
83    !sed -i -e 's/\/data\/librispeech\/librivox-test-clean/test/' {project_name}/conf.py
84    #!sed -i -e 's/# "use_lang/"use_lang/' {project_name}/conf.py
85    !echo 'backend = "librosa"' >> {project_name}/conf.py
86    #!cat {project_name}/conf.py
87    !echo "wav_filename, wav_filesize, transcript" > {project_name}/test.csv
88    !echo "test.wav, UNUSED, UNUSED" >> {project_name}/test.csv
89
90  import sys
91  sys.path.append(project_name)
92  from IPython.display import YouTubeVideo
```

```
93
94  if not exists(join(project_name, 'w2l_log_folder')):
95    import gdown
96    gdown.download('https://drive.google.com/uc?id=1gzGT8HoVNKY1i5HNQTKaSoCu7JHV4siR', 'jasper_10x5
97    !unzip jasper_10x5_dr_sp_nvgrad.zip
98    !mv checkpoint {project_name}/jasper_log_folder
99
100 !cd /content/{project_name} && python run.py --config_file conf.py --mode=infer --infer_output_fi
101
102 answer = pd.read_csv('/content/OpenSeq2Seq/output.txt')
103
104 !unzip jasper_10x5_dr_sp_nvgrad.zip
105
106 import torch
107 quartz_net = torch.load('/content/quartznet15x5_multidataset/JasperEncoder-STEP-243800.pt') 18972
108
109 !wget https://api.ngc.nvidia.com/v2/models/nvidia/jasper_pyt_ckpt_amp/versions/20.10.0/files/nvid
110
111 jasper = torch.load('/content/nvidia_jasper_210205.pt')
112 sum(p.numel() for p in jasper['ema_state_dict'].values())
113
114 jasper['ema_state_dict']
115
116 import string
117 df_val['predicted_transcript'] = answer['predicted_transcript'].tolist()
118 wer(df_val['normalized_transcription'].apply(lambda x:x.lower().translate(str.maketrans('', '', s
119
120 #@title
121 import os
122 from os.path import exists, join, basename, splitext
123 from IPython.display import YouTubeVideo
124
125 if not exists('quartznet15x5_multidataset'):
126   # download the pretrained weights
127   !wget -nc -q --show-progress -O quartznet15x5.zip https://api.ngc.nvidia.com/v2/models/nvidia/m
128   !unzip quartznet15x5.zip && mkdir quartznet15x5_multidataset && mv Jasper* quartznet15x5.yaml q
129
130 import json
131 from ruamel.yaml import YAML
132 import nemo
133 import nemo_asr
134
135 WORK_DIR = "/content/quartznet15x5_multidataset"
136 MODEL_YAML = "/content/quartznet15x5_multidataset/quartznet15x5.yaml"
137 CHECKPOINT_ENCODER = "/content/quartznet15x5_multidataset/JasperEncoder-STEP-243800.pt"
138 CHECKPOINT_DECODER = "/content/quartznet15x5_multidataset/JasperDecoderForCTC-STEP-243800.pt"
139 # Set this to True to enable beam search decoder
140 ENABLE_NGRAM = False
```

```
# This is only necessary if ENABLE_NGRAM = True. Otherwise, set to empty string
LM_PATH = "<PATH_TO_KENLM_BINARY>"

# Read model YAML
yaml = YAML(typ="safe")
with open(MODEL_YAML) as f:
    jasper_model_definition = yaml.load(f)
labels = jasper_model_definition['labels']

# Instantiate necessary Neural Modules
# Note that data layer is missing from here
neural_factory = nemo.core.NeuralModuleFactory(
    placement=nemo.core.DeviceType.GPU,
    backend=nemo.core.Backend.PyTorch)
data_preprocessor = nemo_asr.AudioToMelSpectrogramPreprocessor(factory=neural_factory)
jasper_encoder = nemo_asr.JasperEncoder(
    jasper=jasper_model_definition['JasperEncoder']['jasper'],
    activation=jasper_model_definition['JasperEncoder']['activation'],
    feat_in=jasper_model_definition['AudioToMelSpectrogramPreprocessor']['features'])
jasper_encoder.restore_from(CHECKPOINT_ENCODER, local_rank=0)
jasper_decoder = nemo_asr.JasperDecoderForCTC(
    feat_in=1024,
    num_classes=len(labels))
jasper_decoder.restore_from(CHECKPOINT_DECODER, local_rank=0)
greedy_decoder = nemo_asr.GreedyCTCDecoder()

def wav_to_text(manifest, greedy=True):
    from ruamel.yaml import YAML
    yaml = YAML(typ="safe")
    with open(MODEL_YAML) as f:
        jasper_model_definition = yaml.load(f)
    labels = jasper_model_definition['labels']

    # Instantiate necessary neural modules
    data_layer = nemo_asr.AudioToTextDataLayer(
        shuffle=False,
        manifest_filepath=manifest,
        labels=labels, batch_size=1)

    # Define inference DAG
    audio_signal, audio_signal_len, _, _ = data_layer()
    processed_signal, processed_signal_len = data_preprocessor(
        input_signal=audio_signal,
        length=audio_signal_len)
    encoded, encoded_len = jasper_encoder(audio_signal=processed_signal,
                                          length=processed_signal_len)
    log_probs = jasper_decoder(encoder_output=encoded)
    predictions = greedy_decoder(log_probs=log_probs)
```

```
189
190     if ENABLE_NGRAM:
191         print('Running with beam search')
192         beam_predictions = beam_search_with_lm(
193             log_probs=log_probs, log_probs_length=encoded_len)
194         eval_tensors = [beam_predictions]
195
196     if greedy:
197         eval_tensors = [predictions]
198
199     tensors = neural_factory.infer(tensors=eval_tensors)
200     if greedy:
201         from nemo_asr.helpers import post_process_predictions
202         prediction = post_process_predictions(tensors[0], labels)
203     else:
204         prediction = tensors[0][0][0][0][1]
205     return prediction
206
207 def create_manifest(file_path):
208     # create manifest
209     manifest = dict()
210     manifest['audio_filepath'] = file_path
211     manifest['duration'] = 18000
212     manifest['text'] = 'todo'
213     with open(file_path+".json", 'w') as fout:
214         fout.write(json.dumps(manifest))
215     return file_path+".json"
216
217 !cd /content/{project_name} && python run.py --config_file conf.py --mode=infer --infer_output_fi
218
219 from tqdm.notebook import tqdm
220 predict = [wav_to_text(create_manifest(f'/content/OpenSeq2Seq/{x}')) for x in tqdm(answer['wav_fi
221
222 answer = pd.read_csv('/content/OpenSeq2Seq/output.txt')
223
224 import string
225 df_val['predicted_transcript'] = [ x[0] for x in predict]
226 wer(df_val['normalized_transcription'].apply(lambda x:x.lower().translate(str.maketrans('', '', s
227
228 # The set of characters accepted in the transcription.
229 characters = [x for x in "abcdefghijklmnopqrstuvwxyz'?! "]
230 # Mapping characters to integers
231 char_to_num = keras.layers.StringLookup(vocabulary=characters, oov_token="")
232 # Mapping integers back to original characters
233 num_to_char = keras.layers.StringLookup(
234     vocabulary=char_to_num.get_vocabulary(), oov_token="", invert=True
235 )
236
```

```
print(
    f"The vocabulary is: {char_to_num.get_vocabulary()} "
    f"(size ={char_to_num.vocabulary_size()})"
)

# An integer scalar Tensor. The window length in samples.
frame_length = 256
# An integer scalar Tensor. The number of samples to step.
frame_step = 160
# An integer scalar Tensor. The size of the FFT to apply.
# If not provided, uses the smallest power of 2 enclosing frame_length.
fft_length = 384


def encode_single_sample(wav_file, label):
    ###########################################
    ##  Process the Audio
    ###########################################
    # 1. Read wav file
    file = tf.io.read_file(wavs_path + wav_file + ".wav")
    # 2. Decode the wav file
    audio, _ = tf.audio.decode_wav(file)
    audio = tf.squeeze(audio, axis=-1)
    # 3. Change type to float
    audio = tf.cast(audio, tf.float32)
    # 4. Get the spectrogram
    spectrogram = tf.signal.stft(
        audio, frame_length=frame_length, frame_step=frame_step, fft_length=fft_length
    )
    # 5. We only need the magnitude, which can be derived by applying tf.abs
    spectrogram = tf.abs(spectrogram)
    spectrogram = tf.math.pow(spectrogram, 0.5)
    # 6. normalisation
    means = tf.math.reduce_mean(spectrogram, 1, keepdims=True)
    stddevs = tf.math.reduce_std(spectrogram, 1, keepdims=True)
    spectrogram = (spectrogram - means) / (stddevs + 1e-10)
    ###########################################
    ##  Process the label
    ###########################################
    # 7. Convert label to Lower case
    label = tf.strings.lower(label)
    # 8. Split the label
    label = tf.strings.unicode_split(label, input_encoding="UTF-8")
    # 9. Map the characters in label to numbers
    label = char_to_num(label)
    # 10. Return a dict as our model is expecting two inputs
    return spectrogram, label

```

```
for i in range(10):
    sp, label = encode_single_sample(metadata_df.loc[i, 'file_name'], metadata_df.loc[i, 'normali
    print(sp.shape)

batch_size = 32
# Define the trainig dataset
train_dataset = tf.data.Dataset.from_tensor_slices(
    (list(df_train["file_name"]), list(df_train["normalized_transcription"]))
)
train_dataset = (
    train_dataset.map(encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE)
    .padded_batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)
train_dataset = train_dataset.cache()

# Define the validation dataset
validation_dataset = tf.data.Dataset.from_tensor_slices(
    (list(df_val["file_name"]), list(df_val["normalized_transcription"]))
)
validation_dataset = (
    validation_dataset.map(encode_single_sample, num_parallel_calls=tf.data.AUTOTUNE)
    .padded_batch(batch_size)
    .prefetch(buffer_size=tf.data.AUTOTUNE)
)
validation_dataset = validation_dataset.cache()


fig = plt.figure(figsize=(8, 5))
for batch in train_dataset.take(1):
    spectrogram = batch[0][0].numpy()
    spectrogram = np.array([np.trim_zeros(x) for x in np.transpose(spectrogram)])
    label = batch[1][0]
    # Spectrogram
    label = tf.strings.reduce_join(num_to_char(label)).numpy().decode("utf-8")
    ax = plt.subplot(2, 1, 1)
    ax.imshow(spectrogram, vmax=1)
    ax.set_title(label)
    ax.axis("off")
    # Wav
    file = tf.io.read_file(wavs_path + list(df_train["file_name"])[0] + ".wav")
    audio, _ = tf.audio.decode_wav(file)
    audio = audio.numpy()
    ax = plt.subplot(2, 1, 2)
    plt.plot(audio)
    ax.set_title("Signal Wave")
    ax.set_xlim(0, len(audio))
    display.display(display.Audio(np.transpose(audio), rate=16000))
```

```
plt.show()

def CTCLoss(y_true, y_pred):
    # Compute the training-time loss value
    batch_len = tf.cast(tf.shape(y_true)[0], dtype="int64")
    input_length = tf.cast(tf.shape(y_pred)[1], dtype="int64")
    label_length = tf.cast(tf.shape(y_true)[1], dtype="int64")

    input_length = input_length * tf.ones(shape=(batch_len, 1), dtype="int64")
    label_length = label_length * tf.ones(shape=(batch_len, 1), dtype="int64")

    loss = keras.backend.ctc_batch_cost(y_true, y_pred, input_length, label_length)
    return loss

from tensorflow.keras.layers import Conv1D, MaxPooling1D, Dropout, \
BatchNormalization, GlobalAveragePooling1D, Softmax, Add, Dense, Activation

def build_model(input_dim, output_dim, lstm_layers=5, lstm_units=128):
    """Model similar to 2CNN + Dence"""
    # Model's input
    input_spectrogram = layers.Input((None, input_dim), name="input")
    # Expand the dimension to use 2D CNN.
    x = layers.Reshape((-1, input_dim, 1), name="expand_dim")(input_spectrogram)
    # Convolution layer 1
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 41],
        strides=[2, 2],
        padding="same",
        use_bias=False,
        name="conv_1",
    )(x)
    x = layers.BatchNormalization(name="conv_1_bn")(x)
    x = layers.ReLU(name="conv_1_relu")(x)
    # Convolution layer 2
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 21],
        strides=[1, 2],
        padding="same",
        use_bias=False,
        name="conv_2",
    )(x)
    x = layers.BatchNormalization(name="conv_2_bn")(x)
    x = layers.ReLU(name="conv_2_relu")(x)
    # Reshape the resulted volume to feed the RNNs layers
    x = layers.Reshape((-1, x.shape[-2] * x.shape[-1]))(x)
    # RNN layers
```

```
    for i in range(1, lstm_layers + 1):
        recurrent = layers.GRU(
            units=lstm_units,
            activation="tanh",
            recurrent_activation="sigmoid",
            use_bias=True,
            return_sequences=True,
            # reset_after=True,
            name=f"lstm_{i}",
        )
        # x = layers.Bidirectional(
            # recurrent, name=f"bidirectional_{i}", merge_mode="concat"
        # )(x)
        # if i < lstm_layers:
            # x = layers.Dropout(rate=0.5)(x)
    # Dense layer
    x = layers.Dense(units=lstm_units * 2, name="dense_1")(x)
    x = layers.ReLU(name="dense_1_relu")(x)
    x = layers.Dropout(rate=0.5)(x)
    # Classification layer
    output = layers.Dense(units=output_dim + 1, activation="softmax")(x)
    # Model
    model = keras.Model(input_spectrogram, output, name="2CNN + Dence")
    # Optimizer
    opt = keras.optimizers.Adam(learning_rate=1e-4)
    # Compile the model and return
    model.compile(optimizer=opt, loss=CTCLoss)
    return model


# Get the model
model = build_model(
    input_dim=fft_length // 2 + 1,
    output_dim=char_to_num.vocabulary_size(),
    lstm_units=512,
)
model.summary(line_length=110)


# A utility function to decode the output of the network
def decode_batch_predictions(pred):
    input_len = np.ones(pred.shape[0]) * pred.shape[1]
    # Use greedy search. For complex tasks, you can use beam search
    results = keras.backend.ctc_decode(pred, input_length=input_len, greedy=True)[0][0]
    # Iterate over the results and get back the text
    output_text = []
    for result in results:
        result = tf.strings.reduce_join(num_to_char(result)).numpy().decode("utf-8")
```

```
        output_text.append(result)
    return output_text


# A callback class to output a few transcriptions during training
class CallbackEval(keras.callbacks.Callback):
    """Displays a batch of outputs after every epoch."""

    def __init__(self, dataset):
        super().__init__()
        self.dataset = dataset

    def on_epoch_end(self, epoch: int, logs=None):
        predictions = []
        targets = []
        for batch in self.dataset:
            X, y = batch
            batch_predictions = model.predict(X)
            batch_predictions = decode_batch_predictions(batch_predictions)
            predictions.extend(batch_predictions)
            for label in y:
                label = (
                    tf.strings.reduce_join(num_to_char(label)).numpy().decode("utf-8")
                )
                targets.append(label)
        wer_score = wer(targets, predictions)
        print("-" * 100)
        print(f"Word Error Rate: {wer_score:.4f}")
        print("-" * 100)
        for i in np.random.randint(0, len(predictions), 2):
            print(f"Target    : {targets[i]}")
            print(f"Prediction: {predictions[i]}")
            print("-" * 100)


# Define the number of epochs.
epochs = 50
# Callback function to check transcription on the val set.
validation_callback = CallbackEval(validation_dataset)
# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    callbacks=[validation_callback],
)

def build_model(input_dim, output_dim, lstm_layers=2, lstm_units=128):
```

```
    """Model similar to DeepSpeech. GRU"""
    # Model's input
    input_spectrogram = layers.Input((None, input_dim), name="input")
    # Expand the dimension to use 2D CNN.
    x = layers.Reshape((-1, input_dim, 1), name="expand_dim")(input_spectrogram)
    # Convolution layer 1
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 41],
        strides=[2, 2],
        padding="same",
        use_bias=False,
        name="conv_1",
    )(x)
    x = layers.BatchNormalization(name="conv_1_bn")(x)
    x = layers.ReLU(name="conv_1_relu")(x)
    # Convolution layer 2
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 21],
        strides=[1, 2],
        padding="same",
        use_bias=False,
        name="conv_2",
    )(x)
    x = layers.BatchNormalization(name="conv_2_bn")(x)
    x = layers.ReLU(name="conv_2_relu")(x)
    # Reshape the resulted volume to feed the RNNs layers
    x = layers.Reshape((-1, x.shape[-2] * x.shape[-1]))(x)
    # RNN layers
    for i in range(1, lstm_layers + 1):
        x = layers.GRU(
            units=lstm_units,
            activation="tanh",
            recurrent_activation="sigmoid",
            use_bias=True,
            return_sequences=True,
            # reset_after=True,
            name=f"lstm_{i}",
        )(x)
        if i < lstm_layers:
            x = layers.Dropout(rate=0.5)(x)
    # Dense layer
    x = layers.Dense(units=lstm_units * 2, name="dense_1")(x)
    x = layers.ReLU(name="dense_1_relu")(x)
    x = layers.Dropout(rate=0.5)(x)
    # Classification layer
    output = layers.Dense(units=output_dim + 1, activation="softmax")(x)
```

```
    # Model
    model = keras.Model(input_spectrogram, output, name="DeepSpeech. GRU")
    # Optimizer
    opt = keras.optimizers.Adam(learning_rate=1e-4)
    # Compile the model and return
    model.compile(optimizer=opt, loss=CTCLoss)
    return model


# Get the model
model = build_model(
    input_dim=fft_length // 2 + 1,
    output_dim=char_to_num.vocabulary_size(),
    lstm_units=512,
)
model.summary(line_length=110)

# Define the number of epochs.
epochs = 50
# Callback function to check transcription on the val set.
validation_callback = CallbackEval(validation_dataset)
# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    callbacks=[validation_callback],
)

def build_model(input_dim, output_dim, lstm_layers=2, lstm_units=128):
    """Model similar to DeepSpeech. LSTM"""
    # Model's input
    input_spectrogram = layers.Input((None, input_dim), name="input")
    # Expand the dimension to use 2D CNN.
    x = layers.Reshape((-1, input_dim, 1), name="expand_dim")(input_spectrogram)
    # Convolution layer 1
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 41],
        strides=[2, 2],
        padding="same",
        use_bias=False,
        name="conv_1",
    )(x)
    x = layers.BatchNormalization(name="conv_1_bn")(x)
    x = layers.ReLU(name="conv_1_relu")(x)
    # Convolution layer 2
    x = layers.Conv2D(
```

```
        filters=32,
        kernel_size=[11, 21],
        strides=[1, 2],
        padding="same",
        use_bias=False,
        name="conv_2",
    )(x)
    x = layers.BatchNormalization(name="conv_2_bn")(x)
    x = layers.ReLU(name="conv_2_relu")(x)
    # Reshape the resulted volume to feed the RNNs layers
    x = layers.Reshape((-1, x.shape[-2] * x.shape[-1]))(x)
    # RNN layers
    x = layers.Dense(units=lstm_units * 2, name="dense_1")(x)
    x = layers.ReLU(name="dense_1_relu")(x)
    x = layers.Dropout(rate=0.5)(x)
    for i in range(1, lstm_layers + 1):
        x = layers.LSTM(
            units=lstm_units,
            activation="tanh",
            recurrent_activation="sigmoid",
            use_bias=True,
            return_sequences=True,
            # reset_after=True,
            name=f"lstm_{i}",
        )(x)
        # x = layers.Bidirectional(
            # recurrent, name=f"bidirectional_{i}", merge_mode="concat"
        # )(x)
        if i < lstm_layers:
            x = layers.Dropout(rate=0.5)(x)
    # Dense layer
    x = layers.Dense(units=lstm_units * 2, name="dense_2")(x)
    x = layers.ReLU(name="dense_2_relu")(x)
    x = layers.Dropout(rate=0.5)(x)
    # Classification layer
    output = layers.Dense(units=output_dim + 1, activation="softmax")(x)
    # Model
    model = keras.Model(input_spectrogram, output, name="DeepSpeech. LSTM")
    # Optimizer
    opt = keras.optimizers.Adam(learning_rate=1e-4)
    # Compile the model and return
    model.compile(optimizer=opt, loss=CTCLoss)
    return model


# Get the model
model = build_model(
    input_dim=fft_length // 2 + 1,
```

```
    output_dim=char_to_num.vocabulary_size(),
    lstm_units=512,
)
model.summary(line_length=110)

# Define the number of epochs.
epochs = 50
# Callback function to check transcription on the val set.
validation_callback = CallbackEval(validation_dataset)
# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    callbacks=[validation_callback],
)

def build_model(input_dim, output_dim, rnn_layers=5, rnn_units=128):
    """Model similar to DeepSpeech2. LSTM"""
    # Model's input
    input_spectrogram = layers.Input((None, input_dim), name="input")
    # Expand the dimension to use 2D CNN.
    x = layers.Reshape((-1, input_dim, 1), name="expand_dim")(input_spectrogram)
    # Convolution layer 1
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 41],
        strides=[2, 2],
        padding="same",
        use_bias=False,
        name="conv_1",
    )(x)
    x = layers.BatchNormalization(name="conv_1_bn")(x)
    x = layers.ReLU(name="conv_1_relu")(x)
    # Convolution layer 2
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 21],
        strides=[1, 2],
        padding="same",
        use_bias=False,
        name="conv_2",
    )(x)
    x = layers.BatchNormalization(name="conv_2_bn")(x)
    x = layers.ReLU(name="conv_2_relu")(x)
    # Reshape the resulted volume to feed the RNNs layers
    x = layers.Reshape((-1, x.shape[-2] * x.shape[-1]))(x)
    # RNN layers
```

```
    for i in range(1, rnn_layers + 1):
        recurrent = layers.LSTM(
            units=rnn_units,
            activation="tanh",
            recurrent_activation="sigmoid",
            use_bias=True,
            return_sequences=True,
            reset_after=True,
            name=f"gru_{i}",
        )
        x = layers.Bidirectional(
            recurrent, name=f"bidirectional_{i}", merge_mode="concat"
        )(x)
        if i < rnn_layers:
            x = layers.Dropout(rate=0.5)(x)
    # Dense layer
    x = layers.Dense(units=rnn_units * 2, name="dense_1")(x)
    x = layers.ReLU(name="dense_1_relu")(x)
    x = layers.Dropout(rate=0.5)(x)
    # Classification layer
    output = layers.Dense(units=output_dim + 1, activation="softmax")(x)
    # Model
    model = keras.Model(input_spectrogram, output, name="DeepSpeech2. LSTM")
    # Optimizer
    opt = keras.optimizers.Adam(learning_rate=1e-4)
    # Compile the model and return
    model.compile(optimizer=opt, loss=CTCLoss)
    return model

K.clear_session()

# Get the model
model = build_model(
    input_dim=fft_length // 2 + 1,
    output_dim=char_to_num.vocabulary_size(),
    rnn_units=256,
    rnn_layers = 2
)

model.summary(line_length=110)

# Define the number of epochs.
epochs = 50
# Callback function to check transcription on the val set.
validation_callback = CallbackEval(validation_dataset)
# Train the model
history = model.fit(
    train_dataset,
```

```
    validation_data=validation_dataset,
    epochs=epochs,
    callbacks=[validation_callback],
    verbose = 1
)

def build_model(input_dim, output_dim, rnn_layers=5, rnn_units=128):
    """Model similar to DeepSpeech2. GRU"""
    # Model's input
    input_spectrogram = layers.Input((None, input_dim), name="input")
    # Expand the dimension to use 2D CNN.
    x = layers.Reshape((-1, input_dim, 1), name="expand_dim")(input_spectrogram)
    # Convolution layer 1
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 41],
        strides=[2, 2],
        padding="same",
        use_bias=False,
        name="conv_1",
    )(x)
    x = layers.BatchNormalization(name="conv_1_bn")(x)
    x = layers.ReLU(name="conv_1_relu")(x)
    # Convolution layer 2
    x = layers.Conv2D(
        filters=32,
        kernel_size=[11, 21],
        strides=[1, 2],
        padding="same",
        use_bias=False,
        name="conv_2",
    )(x)
    x = layers.BatchNormalization(name="conv_2_bn")(x)
    x = layers.ReLU(name="conv_2_relu")(x)
    # Reshape the resulted volume to feed the RNNs layers
    x = layers.Reshape((-1, x.shape[-2] * x.shape[-1]))(x)
    # RNN layers
    for i in range(1, rnn_layers + 1):
        recurrent = layers.GRU(
            units=rnn_units,
            activation="tanh",
            recurrent_activation="sigmoid",
            use_bias=True,
            return_sequences=True,
            reset_after=True,
            name=f"gru_{i}",
        )
        x = layers.Bidirectional(
```

```
            recurrent, name=f"bidirectional_{i}", merge_mode="concat"
        )(x)
        if i < rnn_layers:
            x = layers.Dropout(rate=0.5)(x)
    # Dense layer
    x = layers.Dense(units=rnn_units * 2, name="dense_1")(x)
    x = layers.ReLU(name="dense_1_relu")(x)
    x = layers.Dropout(rate=0.5)(x)
    # Classification layer
    output = layers.Dense(units=output_dim + 1, activation="softmax")(x)
    # Model
    model = keras.Model(input_spectrogram, output, name="DeepSpeech2. GRU")
    # Optimizer
    opt = keras.optimizers.Adam(learning_rate=1e-4)
    # Compile the model and return
    model.compile(optimizer=opt, loss=CTCLoss)
    return model

K.clear_session()

# Get the model
model = build_model(
    input_dim=fft_length // 2 + 1,
    output_dim=char_to_num.vocabulary_size(),
    rnn_units=256,
    rnn_layers = 2
)

model.summary(line_length=110)

# Define the number of epochs.
epochs = 50
# Callback function to check transcription on the val set.
validation_callback = CallbackEval(validation_dataset)
# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=epochs,
    callbacks=[validation_callback],
    verbose = 1
)
```