بسمه تعالی

راحله غلامی ، کدملی 2080095579

**Task2: ایجاد یک مدل بازشناسی گفتار برای زبان فارسی بر مبنای مدل jasper در Nemo.**

این مدل را با یک گیگا بایت داده common voice فارسی در کولب آموزش می‌دهیم و کارایی مدل بر مبنای معیار WER روی داده تست common voice فارسی گزارش می‌شود.

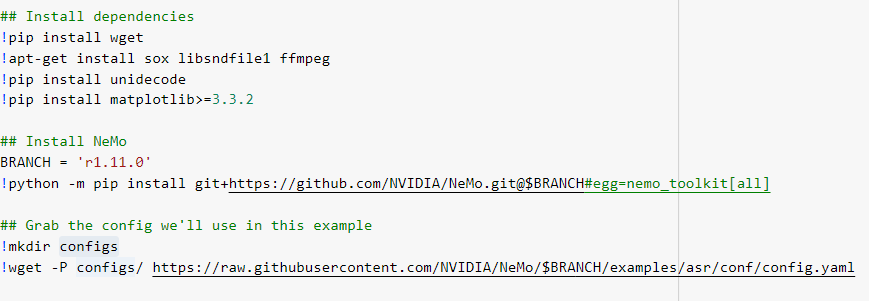
ابتدا با ورود به صفحه زیر از سایت NVIDIA می‌توان به انواع مدل‌های موجود برای بازشناسی گفتار (ASR)، پردازش زبان طبیعی (NLP)، تبدیل متن به گفتار (TTS) دسترسی داشت. البته برای این مساله ما نیازمند مدل ASR هستیم.

https://docs.nvidia.com/deeplearning/nemo/user-guide/docs/en/stable/starthere/tutorials.html

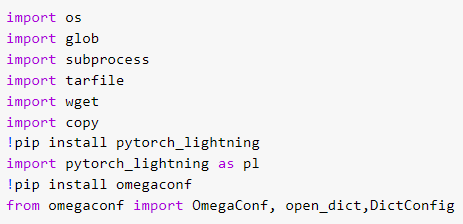
سپس با انتخاب کد ASR with NeMo می‌توان از مدل موجود بهره برد. مدل موردنظر برای آموزش، مدل جاسپر است. به طور خلاصه، معماری جاسپر از یک ساختار بلوک تکرارشونده تشکیل شده است که از کانولوشن‌های یک بُعدی استفاده می‌کند. در یک مدل Jasper\_KxR، تعداد R تا زیربلوک (sub-block) شامل: کانولوشن‌ یک بُعدی، batch norm، ReLU و dropout است که در یک بلوک واحد گروه‌بندی شده و سپس K بار تکرار می‌شوند. همچنین یک بلوک اضافی در ابتدا و چند بلوک دیگر در انتها داریم که نسبت به K و R ثابت هستند و از CTC loss استفاده می‌کنیم. QuartzNet نوع بهتری نسبت به Jasper است با یک تفاوت کلیدی که از کانولوشن‌های یک بُعدی قابل جداسازیِ زمانیِ کانال استفاده می‌کند. در نتیجه به طور چشمگیری تعداد وزن‌ها کاهش یافته و در عین حال دقت مشابهی را حفظ می‌کند.

ما از جعبه ابزار Neural Modules (NeMo) برای این قسمت استفاده خواهیم کرد، ابتدا باید NeMo و وابستگی‌های آن را دانلود و نصب کنیم. NeMo به ما اجازه می‌دهد تا اجزای (ماژول) مدل خود را به راحتی به هم متصل کنیم، مانند لایه داده، لایه‌های میانی و تلفات مختلف، بدون نگرانی بیش از حد در مورد جزئیات پیاده‌سازی قطعات جداگانه یا اتصالات بین ماژول ها. NeMo همچنین دارای مدل‌های کاملی است که فقط به داده‌ها و هایپرپارامترهای ما برای آموزش نیاز دارند.

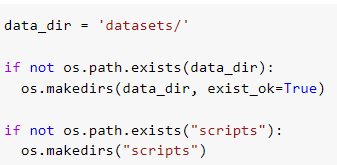
در این کد ابتدا باید پیش‌نیازهای اولیه اجرا شود که شامل نصب یکسری package ها، نصب Nemo و config کردن است که به‌صورت زیر انجام می‌شود:



استفاده از توابع و کتابخانه‌هایی که در ادامه موردنیاز هستند:

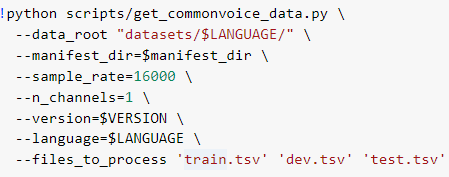


تعریف فولدر dataset برای آنکه سیگنال‌های دانلود شده (سیگنال‌های موجود در دیتاست common voice که در ادامه توسط کولب دانلود می‌شوند) در آن قرار بگیرند:

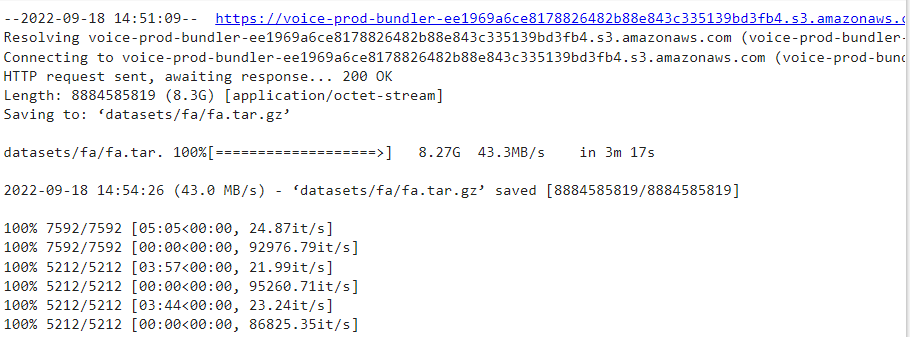


دانلود دیتاست: دیتاست موردنظر همانطور که در قسمت‌های بالاتر نیز گفته شد شامل دیتاست common voice است و از نسخه 6.1 آن و به زبان فارسی استفاده شده. به همین منظور در کد زیر ابتدا مشخصات دیتاست موردنظر نظیر ورژن و زبان موردنظر تعیین شده‌است تا برنامه در ادامه بتواند دیتاست مربوطه را دانلود نماید.





با اجرای کد مربوطه دیتاست موردنظر که حجمی برابر با 8 گیگابایت دانلود می‌شود.



کاری که اکنون باید انجام دهیم این است که برای داده‌های آموزشی و ارزیابی خود مانیفست‌هایی ایجاد کنیم که حاوی ابرداده فایل‌های صوتی ما باشد. مجموعه داده‌های NeMo در قالب مانیفست استاندارد شده‌ای هستند که در آن هر خط، با یک نمونه صدا مطابقت دارد، به طوری که تعداد خطوط در یک مانیفست برابر با تعداد نمونه‌هایی است که توسط آن مانیفست نشان داده می‌شود. یک خط باید شامل: مسیر فایل صوتی، رونوشت متناظر (یا مسیر فایل رونوشت) و مدت زمان نمونه صوتی باشد.



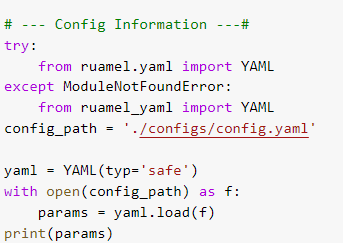
مجموعه ASR NeMo با بسیاری از بلوک‌ها و حتی مدل‌های کامل ارائه می‌شود که می‌توانیم برای آموزش و ارزیابی از آن‌ها استفاده کنیم. علاوه بر این، چندین مدل با وزن‌های از پیش آموزش دیده عرضه می‌شوند. به عنوان مثال یک مدل کامل QuartzNet15x5 قابل استفاده است:

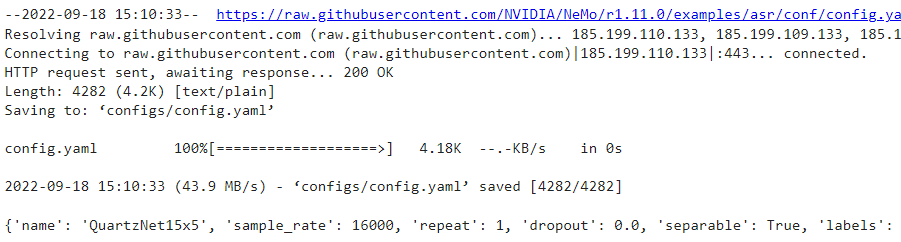


تعیین مدل با یک فایل پیکربندی : YAML

برای این آموزش، یک مدل Jasper\_4x1 با K = 4 بلوک از sub-block (R=1) و greedy CTC با استفاده از پیکربندی موجود در ./configs/config.yaml می‌سازیم. بخش مدل از فایل پیکربندی، شامل توصیف معماری مدل است. مدل حاوی یک ورودی با برچسب encoder، با فیلدی به نام جاسپر است که حاوی لیستی با چندین ورودی است. هر یک از اعضای این لیست یک بلوک را در مدل مشخص می‌کند.

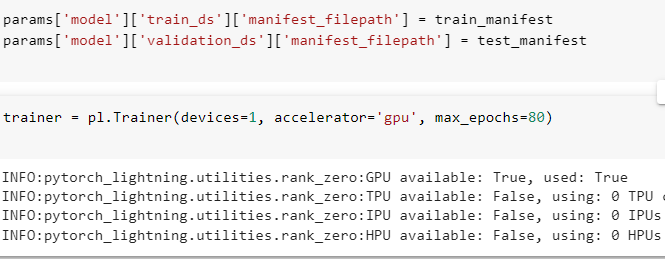
استفاده از یک پیکربندی YAML برای دریافت یک نمای کلی سریع و قابل خواندن از ظاهر معماری مفید است و به ما این امکان را می‌دهد تا مدل را عوض کنیم و پیکربندی‌ها را به راحتی بدون نیاز به تغییر کد اجرا کنیم.

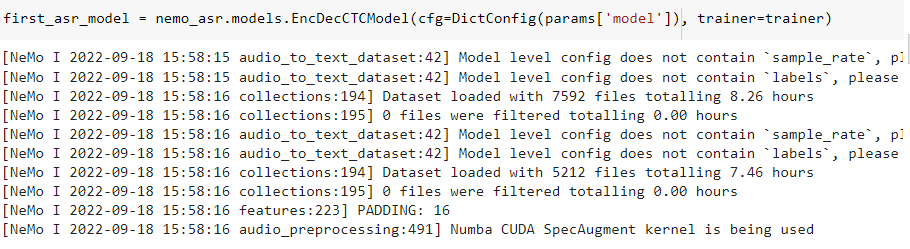




آموزش با کمک pytorch-lightening :

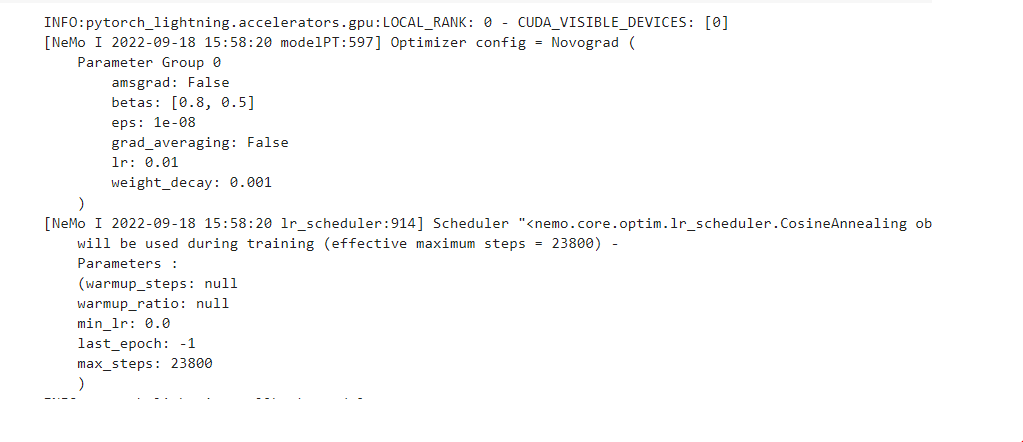
مدل‌های NeMo بر اساس ماژول PytorchLightning هستند و توصیه می‌شود از PytorchLightning برای آموزش و تنظیم دقیق استفاده شود، زیرا استفاده از آموزش با دقت ترکیبی و توزیع شده را بسیار آسان می‌کند. بنابراین برای شروع، Trainer را برای آموزش در GPU برای 80 epoch ایجاد می‌کنیم.

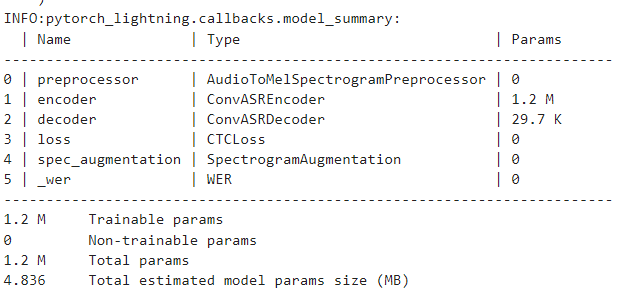




سپس انجام train با کد زیر :

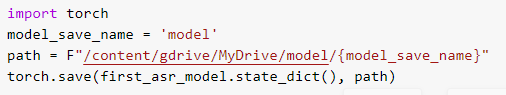




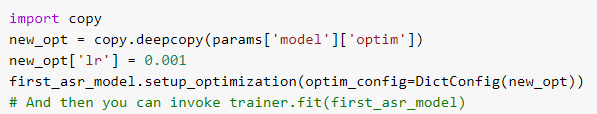


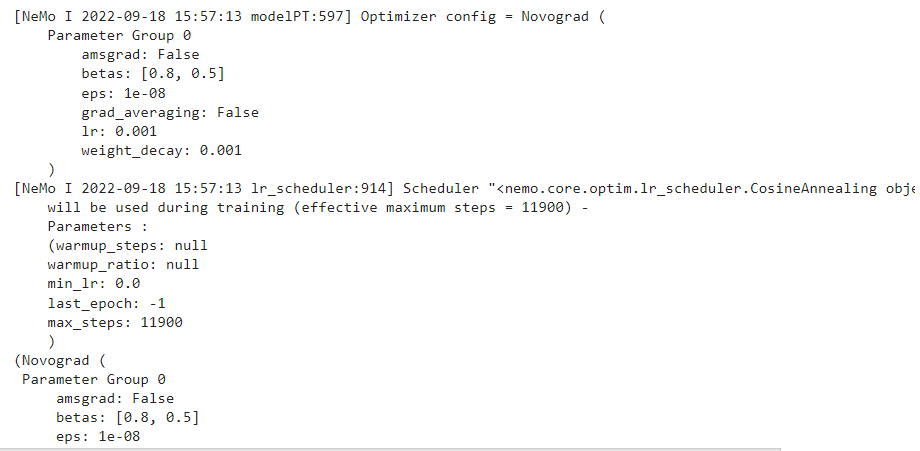


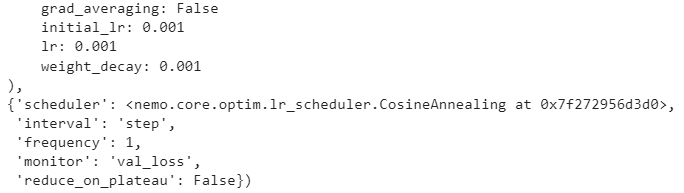
ذخیره مدل در google drive:



می‌توانیم نرخ یادگیری را تغییر دهیم. برای انجام این کار، می‌توان new\_opt dict ایجاد نمود و نرخ یادگیری مورد نظر خود را تنظیم کنیم، سپس  <model>.setup\_optimization()  را با پارامترهای بهینه‌سازی جدید فراخوانی کنیم.







محاسبه‌ی میزان خطا (WER) بین پیش‌بینی ها و مرجع درست :

