```
!pip install torchtext==0.10.1
!pip install matplotlib==3.1.3
```

```
Requirement already satisfied: torchtext==0.10.1 in /usr/local/lib/python3.7/
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: torch==1.9.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: matplotlib==3.1.3 in /usr/local/lib/python3.7/
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /u
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: numpy>=1.11 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-pack
```

```
!pip install wandb
```

```
Requirement already satisfied: wandb in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: yaspin>=1.0.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: six>=1.13.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: GitPython>=1.0.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: shortuuid>=0.5.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: docker-pycreds>=0.4.0 in /usr/local/lib/python
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.7
Requirement already satisfied: Click!=8.0.0,>=7.0 in /usr/local/lib/python3.7
Requirement already satisfied: configparser>=3.8.1 in /usr/local/lib/python3.
Requirement already satisfied: sentry-sdk>=1.0.0 in /usr/local/lib/python3.7/
Requirement already satisfied: promise<3,>=2.0 in /usr/local/lib/python3.7/di
Requirement already satisfied: pathtools in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: protobuf>=3.12.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.7/dist
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/pytho
Requirement already satisfied: subprocess32>=3.5.3 in /usr/local/lib/python3.
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/p
Requirement already satisfied: gitdb<5,>=4.0.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: smmap<6,>=3.0.1 in /usr/local/lib/python3.7/di
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /us
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
Requirement already satisfied: termcolor<2.0.0,>=1.1.0 in /usr/local/lib/pyth
```

```
!pip install torchaudio==0.9.1
```

```
Requirement already satisfied: torchaudio==0.9.1 in /usr/local/lib/python3.7/
Requirement already satisfied: torch==1.9.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/
```

```
"""
You can run either this notebook locally (if you have all the dependencies ar

Instructions for setting up Colab are as follows:
1. Open a new Python 3 notebook.
2. Import this notebook from GitHub (File -> Upload Notebook -> "GITHUB" tab
3. Connect to an instance with a GPU (Runtime -> Change runtime type -> selec
4. Run this cell to set up dependencies.
5. Restart the runtime (Runtime -> Restart Runtime) for any upgraded packages
"""
# If you're using Google Colab and not running locally, run this cell.

## Install dependencies
!pip install wget
!apt-get install sox libsndfile1 ffmpeg
!pip install unidecode


## Install NeMo
BRANCH = 'r1.4.0'
!python -m pip install git+https://github.com/NVIDIA/NeMo.git@$BRANCH#egg=nem

## Grab the config we'll use in this example
!mkdir configs
!wget -P configs/ https://raw.githubusercontent.com/NVIDIA/NeMo/$BRANCH/examp
!pip install matplotlib==3.1.3
"""
Remember to restart the runtime for the kernel to pick up any upgraded packag
Alternatively, you can uncomment the exit() below to crash and restart the ke
that you want to use the "Run All Cells" (or similar) option.
"""
# exit()
```

```
Requirement already satisfied: wget in /usr/local/lib/python3.7/dist-packages
Reading package lists... Done
Building dependency tree
Reading state information... Done
libsndfile1 is already the newest version (1.0.28-4ubuntu0.18.04.2).
ffmpeg is already the newest version (7:3.4.8-0ubuntu0.2).
sox is already the newest version (14.4.2-3ubuntu0.18.04.1).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
Requirement already satisfied: unidecode in /usr/local/lib/python3.7/dist-pac
Collecting nemo_toolkit[all]
  Cloning https://github.com/NVIDIA/NeMo.git (to revision r1.4.0) to /tmp/pip
  Running command git clone -q https://github.com/NVIDIA/NeMo.git /tmp/pip-in
  Running command git checkout -b r1.4.0 --track origin/r1.4.0
  Switched to a new branch 'r1.4.0'
  Branch 'r1.4.0' set up to track remote branch 'r1.4.0' from 'origin'.
Requirement already satisfied: numpy>=1.18.2 in /usr/local/lib/python3.7/dist
Requirement already satisfied: onnx>=1.7.0 in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/di
Requirement already satisfied: torch<1.10,>1.7 in /usr/local/lib/python3.7/di
Requirement already satisfied: wrapt in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: ruamel.yaml in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.7/dist-
Requirement already satisfied: sentencepiece<1.0.0 in /usr/local/lib/python3.
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: numba in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: wget in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: frozendict in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: unidecode in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: black==19.10b0 in /usr/local/lib/python3.7/dis
Requirement already satisfied: isort[requirements]<5 in /usr/local/lib/python
Requirement already satisfied: parameterized in /usr/local/lib/python3.7/dist
Requirement already satisfied: pytest in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pytest-runner in /usr/local/lib/python3.7/dist
Requirement already satisfied: sphinx in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: sphinxcontrib-bibtex in /usr/local/lib/python3
Requirement already satisfied: wandb in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: inflect in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: pypinyin in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: attrdict in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pystoi in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pesq in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: g2p_en in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packag
Requirement already satisfied: pytorch-lightning<1.5,>=1.4.8 in /usr/local/li
Requirement already satisfied: torchmetrics>=0.4.1rc0 in /usr/local/lib/pytho
Requirement already satisfied: transformers>=4.0.1 in /usr/local/lib/python3.
Requirement already satisfied: webdataset<=0.1.62,>=0.1.48 in /usr/local/lib/
Requirement already satisfied: omegaconf>=2.1.0 in /usr/local/lib/python3.7/d
Requirement already satisfied: hydra-core>=1.1.0 in /usr/local/lib/python3.7/
Requirement already satisfied: braceexpand in /usr/local/lib/python3.7/dist-p
Requirement already satisfied: editdistance in /usr/local/lib/python3.7/dist-
Requirement already satisfied: kaldi-io in /usr/local/lib/python3.7/dist-pack
```

```
# Папка, где будет размещаться датасет Golos
data_dir = '/content'
```

```
import glob
import os
import subprocess
import tarfile
import wget
```

```
# Загрузка датасета Golos
def load(golos_url, fname):
  if not os.path.exists(os.path.join(data_dir, fname)):
      golos_path = wget.download(golos_url, os.path.join(data_dir, fname))
      print(f"Dataset downloaded at: {golos_path}")
  else:
      print("Tarfile already exists.")
      golos_path = os.path.join(data_dir, fname)
  return golos_path

golos_path = load("https://sc.link/Kqr", "test.tar")

if not os.path.exists(os.path.join(data_dir, '/test/')):
    tar = tarfile.open(golos_path)
    tar.extractall(path=data_dir)
```

```
    Tarfile already exists.
```

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)
```

```
    Mounted at /content/drive
```

Здесь я распоковала и crowd, и farfield к себе на диск для удобства работы.

```
!tar xvf '/content/drive/MyDrive/универ/golos/golos_opus.tar' train_opus/farf
```

```
len(os.listdir('/content/train_opus/farfield'))
```

```
    124003
```

Теперь у нас есть куча файлов из farfield, можно по репозиторию заметить, что довольно много обучения занимало именно farfield. Но ничего. Проанализируем ошибки и попробуем дообучить.

Теперь в папке `/content` должна размещаться папка `test` с тестовыми данных датасета Golos. Она содержит файлы манифесты `./test/crowd/manifest.jsonl`, `./test/farfield/manifest.jsonl` и аудио файлы в подпапках `./test/crowd/files/` и `./test/farfield/files/`.

Мы можем загрузить и посмотреть данные. Например файл `./test/crowd/files/e632f7d39c15e7edfc665b91e6f2071f.wav` это четырехсекундная запись мужчины, который произносит фразу `Купить моющее средство`. Чтобы убедиться в этом давайте послушаем запись:

Обработаем файлы farfield. Так как на них не было manifest, я собрала его по крупицам сама. Последовательность немного странная, да. Но таким образом мы избавляемся от возможности увидеть файл, который еще не встречался в json файлах, если это может быть.

```
import json


farfield_files_names = os.listdir('/content/train_opus/farfield')
farfield_files_names[:3]
```

```
['230dd47c42e334f5cdb64d840b37d32c.opus',
 '1605d1a07493698d91e31eb04f87305b.opus',
 '64c595dc27d7d7aebd8152c41c070286.opus']
```

```
train_farfield_data = []


for file_path in ['/content/drive/MyDrive/универ/golos/1hour.jsonl', '/conter
                  '/content/drive/MyDrive/универ/golos/10hours.jsonl', '/cont
                  '/content/drive/MyDrive/универ/golos/manifest.jsonl']:
  with open(file_path) as f:
    for line in f:
      file_info = json.loads(line)
      if file_info['audio_filepath'].startswith('farfield'):
        train_farfield_data.append(file_info)

train_farfield_data[:5]
```

```
[{'audio_filepath': 'farfield/63454fe413b6cc91ab8fe29a7877abbe.opus',
  'duration': 2.1576875,
  'id': '63454fe413b6cc91ab8fe29a7877abbe',
  'text': ' '},
 {'audio_filepath': 'farfield/2a96f8c0392f445325cb537d81400ad7.opus',
  'duration': 4.8178125,
  'id': '2a96f8c0392f445325cb537d81400ad7',
  'text': 'джой включить даб степ'},
 {'audio_filepath': 'farfield/b54c690dcc83803bcd799ab38966fa83.opus',
  'duration': 4.08,
  'id': 'b54c690dcc83803bcd799ab38966fa83',
  'text': 'потерял кошелек в нем была карта сбербанк'},
 {'audio_filepath': 'farfield/f8628b71c6a8d8bceb86bab60bc36bef.opus',
  'duration': 2.2026875,
  'id': 'f8628b71c6a8d8bceb86bab60bc36bef',
  'text': 'три восемьсот евро'},
 {'audio_filepath': 'farfield/141a61eb8fc5eb52ef1c5fc585d6ad87.opus',
  'duration': 3.15625,
  'id': '141a61eb8fc5eb52ef1c5fc585d6ad87',
  'text': ' '}]
```

```
names = farfield_files_names
farfield_data = []


for info in train_farfield_data:
```

```
  name = info['audio_filepath'].split('/')[1]
  if name in names:
    names.remove(name)
    farfield_data.append(info)
```

```
len(farfield_data)
```

```
116423
```

```
len(train_farfield_data)
```

```
142943
```

Гипотеза подветрдилась, где-то что-то потерялось. Но ничего страшного :)

```
for file_info in farfield_data:
  file_info['audio_filepath'] = '/content/drive/MyDrive/train_opus/' + file_i
```

```
with open('/content/drive/MyDrive/универ/golos/farfield_train.json', "w", enc
    for info in farfield_data:
      json.dump(info, outfile, ensure_ascii=False)
      outfile.write('\n')
```

```
!head /content/drive/MyDrive/универ/golos/farfield_train.json
```

```
{"id": "99a43331e70eda058820ba457f40dac1", "audio_filepath": "/content/drive/
{"id": "eac60f97c53ef63d1a64f4973bb356bf", "audio_filepath": "/content/drive/
{"id": "992fd03a868431004a52eec5feb5dcec", "audio_filepath": "/content/drive/
{"id": "1f250760d9dda6a7e7f0a195e42b50c5", "audio_filepath": "/content/drive/
{"id": "518a94545125262bd397729016e90536", "audio_filepath": "/content/drive/
{"id": "6884b1322ac8122e84f984df771412ee", "audio_filepath": "/content/drive/
{"id": "0de10551f946453a9d018da179cd99ea", "audio_filepath": "/content/drive/
{"id": "49cb9a370b7f173a844ade5466d4b3f0", "audio_filepath": "/content/drive/
{"id": "11c6aecd035716c96c745404a70611ee", "audio_filepath": "/content/drive/
{"id": "df85c3bd12e87386f762c2a50f11b2c0", "audio_filepath": "/content/drive/
```

Сбор по крупицам закончен.

Использование NeMo для распонавания речи

Теперь мы знаем что такое задача автоматичского распознавания речи и речевые данные, давайте использовать NeMo для распознавания речи.

Мы будем использовать **Neural Modules (NeMo) toolkit**, поэтому нужно скачать и установить все ее зависимости. Для этого следуйте инструкции в репозитории GitHub page, или документации documentation.

NeMo позволяет нам легко использовать все необходимые компоненты для нашей модели: dataloader, промежуточные сверточные или рекурентные слои, разные loss функции без необходимости разбираться в деталях реализации разных моедлей. В NeMo содержатся готовые реализованные модели в которых достаточно подать свои данные и задать гиперпараметры для обучения.

```
# NeMo's "core" package
import nemo
# NeMo's ASR collection - this collections contains complete ASR models and
# building blocks (modules) for ASR
import nemo.collections.asr as nemo_asr
```

```
[NeMo W 2021-11-30 19:52:37 optimizers:47] Apex was not found. Using the lamb
################################################################################
### WARNING, path does not exist: KALDI_ROOT=/mnt/matylda5/iveselyk/Tools/kal
###          (please add 'export KALDI_ROOT=<your_path>' in your $HOME/.profi
###          (or run as: KALDI_ROOT=<your_path> python <your_script>.py)
################################################################################

[NeMo W 2021-11-30 19:52:38 experimental:28] Module <class 'nemo.collections.
```

Исползование предобученной модели

Коллекция для распознавания речи в NeMo содержит готовые блоки, которые можно использовать чтобы тренировать и использовать свою модель. Кроме этого существет ряд предобученных моделей, которые можно просто скачать и исползовать. Давайте скачаем и инициализируем готовую модель QuartzNet15x5, обученную на открытом датасете Golos.

```
load("https://sc.link/ZMv", "QuartzNet15x5_golos.nemo")

asr_model = nemo_asr.models.EncDecCTCModel.restore_from(os.path.join(data_dir
```

Теперь указваем список фалов которые мы хотим транскрибировать и передаем в нашу модель. Это будет работать для относительно коротких аудио (<25 секунд) файлов.

Обучение с нуля

Для обучения нужно подготовить данные в нужном формате. Для этого добавим абсолютные пути к относительным в наших манифестах и используем их для обучения.

```
# --- Building Manifest Files --- #
import json

# Function to build a manifest
```

```
# Function to build a manifest
def build_manifest(manifest_rel, manifest_abs):
    manifest_path = os.path.split(os.path.abspath(manifest_rel))[0]
    with open(manifest_rel, 'r') as fin:
        with open(manifest_abs, 'w') as fout:
            for line in fin:
                metadata = json.loads(line)
                metadata["audio_filepath"]=os.path.join(manifest_path, metada
                json.dump(metadata, fout)
                fout.write('\n')

# Building Manifests
print("******")
train_rel = os.path.join(data_dir, 'test/farfield/manifest.jsonl')
train_abs = os.path.join(data_dir, 'test/farfield/farfield.jsonl')
if not os.path.isfile(train_abs):
  build_manifest(train_rel, train_abs)
test_manifest = train_abs
print("test_manifest", test_manifest)

train_rel = os.path.join(data_dir, 'test/crowd/manifest.jsonl')
train_abs = os.path.join(data_dir, 'test/crowd/crowd.jsonl')
if not os.path.isfile(train_abs):
  build_manifest(train_rel, train_abs)
train_manifest = train_abs
print("train_manifest", train_manifest)
```

```
******
test_manifest /content/test/farfield/farfield.jsonl
train_manifest /content/test/crowd/crowd.jsonl
```

Тут сделаем трюк. Как и было сказано выше, лучше модель работает с короткими аудио. Возьмем только аудио короче 10 секунд, а также аудио, у которых есть транскрибция, а есть такие у которых ее нет)

```
unused, used = 0, 0


with open('test/crowd/manifest.jsonl') as read_file:
  data = [json.loads(jline) for jline in read_file.read().splitlines()]

with open('test/crowd/manifest.json', 'w') as write_file:
  for line in data:
    if line["duration"] < 10 and line['text']:
      line['audio_filepath'] = '/content/test/crowd/' + line['audio_filepath'
      json.dump(line, write_file, ensure_ascii=False)
      write_file.write('\n')
      used += 1
    else:
      unused += 1
!head test/crowd/manifest.json
```

```
{"id": "e632f7d39c15e7edfc665b91e6f2071f", "audio_filepath": "/content/test/c
{"id": "5db5df8bb9e3b6660b2a04b34d4a355d", "audio_filepath": "/content/test/c
{"id": "2c471aedc6979109f28cd53c58f8c4fb", "audio_filepath": "/content/test/c
{"id": "756a137ee9debde4a008adc4a4121dc7", "audio_filepath": "/content/test/c
{"id": "1ee3b00170123a6723a40e129b2f6bce", "audio_filepath": "/content/test/c
{"id": "35e8b07c1109b98209aded328fa6da0e", "audio_filepath": "/content/test/c
{"id": "c35deaf94921ca67be23c19580a13397", "audio_filepath": "/content/test/c
{"id": "769fc2582dd82d1a3d4b7de9154d43b4", "audio_filepath": "/content/test/c
{"id": "6a5c6ed155b43cdc77d2225084f6bd0a", "audio_filepath": "/content/test/c
{"id": "38bf4f3c89e0507ee921a669fec12e9d", "audio_filepath": "/content/test/c
```

```
print(f'used {used} and unused {unused}')
```

```
used 9878 and unused 116
```

То же самое сделаем с farfield.

```
unused, used = 0, 0


with open('/content/drive/MyDrive/универ/golos/farfield_train.json') as read_
  data = [json.loads(jline) for jline in read_file.read().splitlines()]

with open('/content/drive/MyDrive/универ/golos/farfield_train_clean.json', 'w
  for line in data:
    if line["duration"] < 10 and line['text']:
      json.dump(line, write_file, ensure_ascii=False)
      write_file.write('\n')
      used += 1
    else:
      unused += 1
!head /content/drive/MyDrive/универ/golos/farfield_train_clean.json
```

```
{"id": "99a43331e70eda058820ba457f40dac1", "audio_filepath": "/content/drive/
{"id": "eac60f97c53ef63d1a64f4973bb356bf", "audio_filepath": "/content/drive/
{"id": "992fd03a868431004a52eec5feb5dcec", "audio_filepath": "/content/drive/
{"id": "1f250760d9dda6a7e7f0a195e42b50c5", "audio_filepath": "/content/drive/
{"id": "518a94545125262bd397729016e90536", "audio_filepath": "/content/drive/
{"id": "6884b1322ac8122e84f984df771412ee", "audio_filepath": "/content/drive/
{"id": "0de10551f946453a9d018da179cd99ea", "audio_filepath": "/content/drive/
{"id": "49cb9a370b7f173a844ade5466d4b3f0", "audio_filepath": "/content/drive/
{"id": "11c6aecd035716c96c745404a70611ee", "audio_filepath": "/content/drive/
{"id": "df85c3bd12e87386f762c2a50f11b2c0", "audio_filepath": "/content/drive/
```

```
print(f'used {used} and unused {unused}')
```

```
used 115535 and unused 888
```

Избавились почти от 1000 файлов в farfield, уже что-то.

Задаем модель при помощи YAML конфиг файла

Для обучения мы создадим модель *Jasper_4x1*, в которой будет `K=4` блоков, один (`R=1`) под-блок и декодер *greedy CTC*, используя конфиг файл в `./configs/config.yaml`.

Ниже приведен конфиг файл, давайте рассмотрим его и найдем части описанной архитектуры Jasper. Модель (model) содержит поле под названием `encoder` с под-полем `jasper` который состоит из списка полей. Каждое поле в списке задает конфигурацию блока в нашей моделе. Каждый блок выглядит примерно так:

```
 - filters: 128
   repeat: 1
   kernel: [11]
   stride: [2]
   dilation: [1]
   dropout: 0.2
   residual: false
   separable: true
   se: true
   se_context_size: -1
```

Первый элемент в спике соответствует первому блоку в Jasper архитектуре.

Параметры обучающего и тестового датасета в полях (`train_ds`) и (`validation_ds`)

Конфиг в формате YAML позволяем легко и в читаемой форме читать и модифицировать модель без необходимости менять код прроограммы.

```
! cat ./configs/config.yaml
```

```
# --- Config Information ---#
try:
    from ruamel.yaml import YAML
except ModuleNotFoundError:
    from ruamel_yaml import YAML
config_path = './configs/config.yaml'

yaml = YAML(typ='safe')
with open(config_path) as f:
    params = yaml.load(f)

print(params)
```

```
    {'name': 'QuartzNet15x5', 'sample_rate': 16000, 'repeat': 1, 'dropout': 0.0,
```

```
# Use Russian vocabulary
print(asr_model.decoder.vocabulary)
params["labels"] = asr_model.decoder.vocabulary
params["model"]["train_ds"]["labels"] = asr_model.decoder.vocabulary
params["model"]["validation_ds"]["labels"] = asr_model.decoder.vocabulary
params["model"]["decoder"]["vocabulary"] = asr_model.decoder.vocabulary
params["model"]["decoder"]["num_classes"] = len(asr_model.decoder.vocabulary)
print(params)
```

```
[' ', 'а', 'б', 'в', 'г', 'д', 'е', 'ж', 'з', 'и', 'й', 'к', 'л', 'м', 'н', '
{'name': 'QuartzNet15x5', 'sample_rate': 16000, 'repeat': 1, 'dropout': 0.0,
```

```
import numpy as np


params['trainer']['max_epochs'] = 10
params['model']['optim']['lr'] = 5*np.e - 5
```

Использование PyTorch Lightning

NeMo модели и модули могуть исползоваться в любом PyTorch проекте где ожидается тип torch.nn.Module.

Однако, NeMo модели созданы на основе PytorchLightning's LightningModule, поэтому рекомендуется использовать PytorchLightning для обучения и дообучения (fine-tuning) так как это позволяет легко применять mixed precision и распределенное обучение. Давайте создадим объект Trainer для обучения на GPU 5 эпох.

```
import pytorch_lightning as pl
trainer = pl.Trainer(gpus=1, max_epochs=5)
```

```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
```

Зададим дополнительно batch size.

```
train_manifest = 'test/crowd/manifest.json'


from omegaconf import DictConfig
params['model']['train_ds']['manifest_filepath'] = train_manifest
params['model']['validation_ds']['manifest_filepath'] = test_manifest
params['model']['train_ds']['batch_size'] = 64
params['model']['validation_ds']['batch_size'] = 64
first_asr_model = nemo_asr.models.EncDecCTCModel(cfg=DictConfig(params['model
```

```
[NeMo I 2021-11-30 20:13:52 audio_to_text_dataset:37] Model level config does
```

```
[NeMo I 2021-11-30 20:13:52 audio_to_text_dataset:37] Model level config does
[NeMo I 2021-11-30 20:13:52 collections:173] Dataset loaded with 9878 files t
[NeMo I 2021-11-30 20:13:52 collections:174] 0 files were filtered totalling
[NeMo I 2021-11-30 20:13:52 audio_to_text_dataset:37] Model level config does
[NeMo I 2021-11-30 20:13:52 audio_to_text_dataset:37] Model level config does
[NeMo I 2021-11-30 20:13:52 collections:173] Dataset loaded with 1916 files t
[NeMo I 2021-11-30 20:13:52 collections:174] 0 files were filtered totalling
[NeMo I 2021-11-30 20:13:52 features:262] PADDING: 16
[NeMo I 2021-11-30 20:13:52 features:279] STFT using torch
```

```
# Use the smaller learning rate we set before
first_asr_model.setup_optimization(optim_config=DictConfig(new_opt))

# Point to the data we'll use for fine-tuning as the training set
first_asr_model.setup_training_data(train_data_config=params['model']['train_

# Point to the new validation data for fine-tuning
first_asr_model.setup_validation_data(val_data_config=params['model']['valida

# And now we can create a PyTorch Lightning trainer and call `fit` again.
trainer = pl.Trainer(gpus=1, max_epochs=5)
trainer.fit(first_asr_model)
```

```
    )
[NeMo I 2021-11-30 20:14:03 audio_to_text_dataset:37] Model level config does
[NeMo I 2021-11-30 20:14:03 audio_to_text_dataset:37] Model level config does
[NeMo I 2021-11-30 20:14:04 collections:173] Dataset loaded with 9878 files t
[NeMo I 2021-11-30 20:14:04 collections:174] 0 files were filtered totalling
[NeMo I 2021-11-30 20:14:04 audio_to_text_dataset:37] Model level config does
[NeMo I 2021-11-30 20:14:04 audio_to_text_dataset:37] Model level config does
[NeMo I 2021-11-30 20:14:04 collections:173] Dataset loaded with 1916 files t
[NeMo I 2021-11-30 20:14:04 collections:174] 0 files were filtered totalling
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
[NeMo I 2021-11-30 20:14:04 modelPT:544] Optimizer config = Novograd (
    Parameter Group 0
        amsgrad: False
        betas: [0.8, 0.5]
        eps: 1e-08
        grad_averaging: False
        lr: 5e-05
        weight_decay: 0.001
    )
[NeMo I 2021-11-30 20:14:04 lr_scheduler:625] Scheduler "<nemo.core.optim.lr_
    will be used during training (effective maximum steps = 775) -
    Parameters :
    (warmup_steps: null
    warmup_ratio: null
    min_lr: 0.0
    last_epoch: -1
    max_steps: 775
    )

  | Name              | Type                              | Params
-------------------------------------------------------------------------
0 | preprocessor      | AudioToMelSpectrogramPreprocessor | 0
1 | encoder           | ConvASREncoder                    | 1.2 M
2 | decoder           | ConvASRDecoder                    | 34.9 K
3 | loss              | CTCLoss                           | 0
4 | spec_augmentation | SpectrogramAugmentation           | 0
5 | _wer              | WER                               | 0
-------------------------------------------------------------------------
1.2 M     Trainable params
0         Non-trainable params
1.2 M     Total params
4.856     Total estimated model params size (MB)
```

Мы можем начать обучение всего одной строчкой!

Let's say we wanted to change the learning rate. To do so, we can create a `new_opt` dict and set our desired learning rate, then call `<model>.setup_optimization()` with the new optimization parameters.

Epoch 4: 100% 185/185 [01:00<00:00, 3.07it/s, loss=807, v_num=5]

```
import copy
new_opt = copy.deepcopy(params['model']['optim'])
new opt['lr'] = 5e-5
```