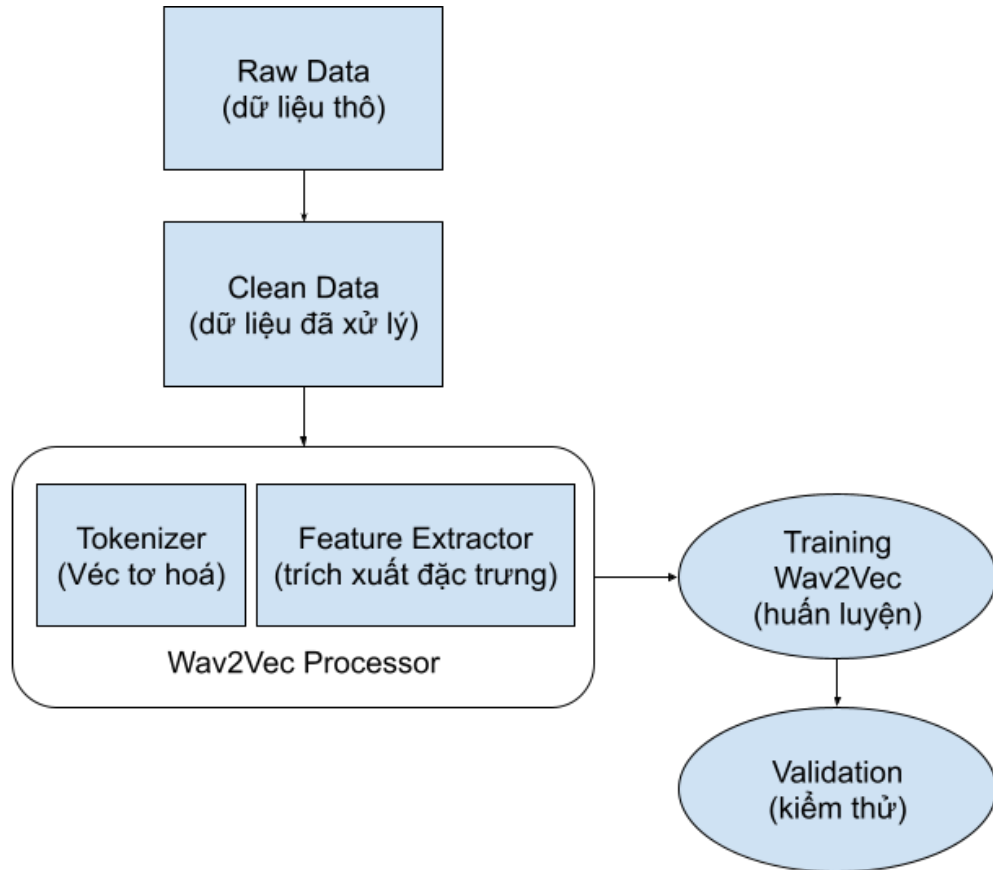


CHƯƠNG 3: XÂY DỰNG MÔ HÌNH

3.1. Chuẩn bị dữ liệu

Trước khi tiến hành chúng ta nên nhìn vào bức tranh toàn cảnh để dễ hình dung toàn bộ các bước như sau:



Hình 7: Quy trình xây dựng mô hình nhận diện giọng nói Wav2Vec

Bộ dữ liệu âm thanh được sử dụng ta sẽ trích xuất từ bộ **common-voice** được cung cấp bởi Mozilla Foundation [5]. Đây là một dự án nguồn mở, nơi mọi người có thể đóng góp giọng nói của họ bằng cách đọc một câu cố định hoặc thời gian nhất định, điều này đặc biệt quan trọng để xác minh xem một tệp âm thanh cụ thể có phù hợp với phiên âm tương ứng hay không. Âm thanh được phát hành dưới dạng tệp MPEG-3 16bit, đơn kênh với tốc độ lấy mẫu 48kHz [45], chất lượng âm thanh phù hợp cho các ứng dụng giọng nói. Bộ dữ liệu này là một bộ dữ liệu âm thanh đa ngôn ngữ tuy nhiên ta sẽ chỉ trích xuất ra Tiếng Việt để sử dụng. Bao gồm tập training (2.53k), test (1.24k), validation (248). Mỗi một điểm dữ liệu sẽ có cấu trúc như sau:

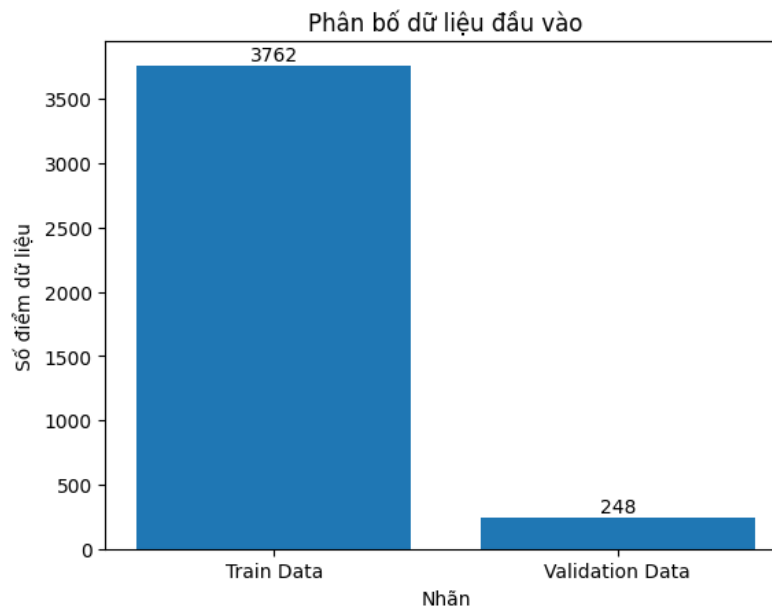
client_id	'c14ad590de005be3a512e187900c5ad5c76921...'
path	'/root/.cache/huggingface/datasets/downloads/extracted/...'
audio	{'path': '/root/.cache/huggingface/datasets/downloads/extracted...}'
sentence	'quả nhiên trúng tuyển vào trường Quốc Lập'
up_votes	2
down_votes	0
age	"
gender	"
accent	"
locale	'vi'
segment	"

Bảng 2: Mô tả một điểm dữ liệu

Ở bài toán này, ta chỉ quan tâm tới 3 thuộc tính là **path**, **audio** và **sentence**. Các thuộc tính còn lại không có giá trị sử dụng cho bài toán nên ta sẽ loại bỏ đi để tiết kiệm bộ nhớ. Do dữ liệu được chia ở tập test có số lượng khoảng một nửa tập training, cho nên để tối ưu nhất ta sẽ gộp tập train và test làm 1 rồi dùng validation để kiểm thử ở bước cuối cùng như kiểm tra sự dao động của độ đo sai số trong quá trình huấn luyện.

```
# Lấy dữ liệu common_voice tiếng Việt
ds_train = load_dataset("mozilla-
foundation/common_voice_11_0", "vi", split="train+test")
ds_test = load_dataset("mozilla-
foundation/common_voice_11_0", "vi", split="validation")
```

Việc kết hợp này được áp dụng do data hiện có quá ít. Và trong quá trình thực nghiệm chúng tôi nhận thấy khi kết hợp tập test vào bộ training độ chính xác của mô hình tăng đáng kể so với việc không chia hoặc chia theo tỉ lệ 80/20. Cuối cùng ta sẽ có phân bố dữ liệu như sau.



Hình 8: Phân bố dữ liệu đầu vào

3.2. Xử lý dữ liệu

Đối với dữ liệu âm thanh (audio) ở tập này dữ liệu âm thanh đang được để Sampling Rate là **48.000 mẫu/1 giây**.

Như đã biết thì Sampling Rate càng cao thì số lượng mẫu âm thanh lưu trữ càng nhiều, nhờ vậy mà âm thanh trở nên trong trẻo, rõ ràng và chính xác với thực tế hơn. Tuy vậy để tối ưu trong học máy ta không cần đến mức đó, với bài toán này ta chỉ cần mức Sampling Rate nhỏ vừa đủ để có thể nghe, với lại khi chỉnh nhỏ giúp mô hình huấn luyện nhanh hơn và giảm chi phí tính toán. Vì vậy ta sẽ dùng hàm **cast_column** để chuyển audio có sampling rate từ 48.000 về 16.000.

Sau khi đã xử lý xong trên cột âm thanh ta sẽ tiếp tục xử lý trên dữ liệu văn bản (**sentence**). Ta nhận ra rằng, từ âm thanh chuyển sang văn bản, những dấu câu thường sẽ không có ý nghĩa, vì đơn giản dấu câu dùng để thể hiện trong văn viết, với lại khi thêm dấu câu mô hình sẽ rất khó có thể học được và xác định nó, nếu một vài dấu câu sai cũng sẽ gây rất nhiều phiền toái về ngữ nghĩa làm người đọc hiểu sai. Vì vậy để đơn giản chúng ta sẽ xóa hết những dấu câu này cũng như những ký tự đặc biệt.

```
# Xóa dấu và các ký tự đặc biệt
import re
chars_to_ignore_regex = '[\, \? \. \! \- \; \: \" \' \% \' \' \? ]'
```

```
def remove_special_characters(batch):  
    batch["sentence"] = re.sub(chars_to_ignore_regex, '',  
    batch["sentence"]).lower() + " "  
    return batch
```

Vậy là chúng ta đã xong các bước xử lý dữ liệu cơ bản, tiếp theo chúng ta sẽ tiến hành xây dựng bộ Tokenizer.

3.3. Xây dựng bộ tokenizer

Máy tính không thể hiểu được các văn bản, ta cần chuyển đổi các từ trong văn bản này sang dạng số (encoding) để máy tính có thể tính được xác suất của từng từ một. Thông thường ta có thể làm nhanh bước này bằng cách sử dụng một bộ thư viện có sẵn để chuyển đổi, tuy nhiên ta sẽ làm thủ công để có thể hiểu rõ hơn cách nó hoạt động, với lại đối với bài toán liên quan tới âm thanh, bộ Tokenizer này sẽ đặc biệt hơn một chút.

```
# Lấy ra tất cả ký tự có thể tìm được trong dataset  
def extract_all_chars(batch):  
    all_text = " ".join(batch["sentence"])  
    vocab = list(set(all_text))  
    return {"vocab": [vocab], "all_text": [all_text]}
```

Ta sẽ tiến hành nối tất cả các hàng trong cột sentence thành một hàng văn bản duy nhất rồi sẽ trích xuất các ký tự xuất hiện trong văn bản đó ra. Lưu ý ở đây ta sẽ chỉ lấy ký tự chứ không lấy từ vựng như các bộ tokenizer LLM thông thường. Vì âm thanh không cần dự đoán ngữ nghĩa nhiều, mô hình chỉ cần xác định đúng từ vựng được nói là được, CTC là một phương pháp học máy thường được sử dụng trong các nhiệm vụ như nhận dạng tiếng nói và nhận dạng văn bản. Nó cho phép mô hình dự đoán chuỗi ký tự mà không cần cung cấp độ dài chuỗi cố định. Trong học máy, việc xử lý dữ liệu một cách đúng đắn rất quan trọng để đảm bảo mô hình được đào tạo hiệu quả, đây được gọi là cơ chế CTC. Việc lấy theo ký tự này giúp ta có thể trích lọc ra gần như tất cả các ký tự chữ cái xuất hiện trong Tiếng Việt hiện nay.

```
| {'ứ': 0,
  'e': 1,
  'ồ': 2,
  'y': 3,
  'ă': 4,
  'z': 5,
  's': 6,
  'à': 7,
  'l': 8,
  'à': 9,
  'ý': 10,
  'ỳ': 11,
  'ạ': 12,
  'ú': 13,
  'ớ': 14,
  'à': 15,
  'ỷ': 16,
  'í': 17,
  'á': 18,
  'ỹ': 19,
```

Hình 9: Danh sách một vài ký tự sau khi lấy được

Việc lấy ký tự cũng giúp ta chỉ cần lưu trữ một lượng nhỏ **94 phần tử** so với số lượng từ vựng đồ sộ trong Tiếng Việt thì lưu ký tự sẽ tối ưu hơn rất nhiều. Tuy nhiên ta cũng sẽ không quên thêm 2 phần tử là [UNK] và [PAD] để cho từ không xác định và padding.

```
# Thêm [UNK] cho từ không xác định
vocab_dict["[UNK]"] = len(vocab_dict)
# Thêm [PAD] để padding cho cùng chiều dài input
vocab_dict["[PAD]"] = len(vocab_dict)
# In chiều dài từ điển
len(vocab_dict)
```

Sau khi xong ta sẽ lưu lại bộ vocab này với định dạng json và tạo tokenizer theo cơ chế CTC mặc định với mô hình Wav2Vec2. Để tạo được bộ tokenizer ta sẽ cần load config từ mô hình gốc, tại đây ta sẽ lấy mô hình **facebook/wav2vec2-large-xlsr-53** và đặt tên cho mô mới là **wav2vec2-vi**.

```
# Tên model để load từ HuggingFace
# Model gốc
model_checkpoint = "facebook/wav2vec2-large-xlsr-53"
# Model đã training trước đó từ model gốc
model_checkpoint = "phatjk/wav2vec2-large-vi"
```

Ta sẽ dùng hàm AutoConfig để lấy các thông tin cài đặt mặc định từ mô hình gốc về nhằm cài lên tokenizer của Tiếng Việt

```
from transformers import AutoConfig
# Load file config.json
config = AutoConfig.from_pretrained(model_wav2vec2)

tokenizer_type = config.model_type if
config.tokenizer_class is None else None
config = config if config.tokenizer_class is not None else
None
```

Cuối cùng ta sẽ gán các giá trị này vào hàm tạo tokenizer của Wav2Vec2.

```
# Tạo bộ tokenizer từ vocab.json
from transformers import Wav2Vec2CTCTokenizer
tokenizer = Wav2Vec2CTCTokenizer.from_pretrained(
    "./",
    config=config,
    tokenizer_type=tokenizer_type,
    unk_token="[UNK]",
    pad_token="[PAD]",
    word_delimiter_token="|",
)
```

3.4. Trích xuất đặc trưng

Đối với trích xuất đặc trưng chúng ta sẽ sử dụng hàm trích xuất mặc định của Wav2Vec, đây là hàm đã được tối ưu hoá để trích lọc đặc trưng dữ liệu cho mô hình Wav2Vec2.

```
# Load Extractor
from transformers import AutoFeatureExtractor
feature_extractor =
AutoFeatureExtractor.from_pretrained(model_wav2vec2)
feature_extractor

Wav2Vec2FeatureExtractor {
  "do_normalize": true,
  "feature_extractor_type": "Wav2Vec2FeatureExtractor",
  "feature_size": 1,
  "padding_side": "right",
  "padding_value": 0,
  "return_attention_mask": true,
  "sampling_rate": 16000 }
```

Lưu ý ở đây các thông số cho hàm trích xuất đặc trưng chúng ta sẽ để mặc định như model gốc padding bên phải của mảng và các thông số còn lại được mô tả như trên.

3.5. Tạo Processor cho Wav2Vec

Sau khi đã tạo xong **tokenizer** và **feature_extractor**, ta sẽ đưa chúng vào hàm **Wav2Vec2Processor** để tạo bộ xử lý cho dữ liệu.

```
# Load Wav2Vec2Processor
from transformers import Wav2Vec2Processor
processor =
Wav2Vec2Processor(feature_extractor=feature_extractor,
tokenizer=tokenizer)
```

Sau khi đã có bộ xử lý ta sẽ tiến hành áp dụng hàm biến đổi này lên toàn bộ các điểm dữ liệu, trong đó ở từng dòng ta sẽ biến đổi *đầu vào* và *đầu ra* của phần tử, với âm thanh đầu vào hàm processor sẽ trích xuất đặc trưng và biến đổi, với văn bản đầu ra thì sẽ tokenizer.

```
# Hàm map processor lên audio và nhãn của từng điểm dữ liệu
trên tập
def prepare_dataset(batch):
    audio = batch["audio"]

    # batched output is "un-batched"
    batch["input_values"] = processor(audio["array"],
        sampling_rate=audio["sampling_rate"]).input_values[0]
    batch["input_length"] = len(batch["input_values"])

    with processor.as_target_processor():
        batch["labels"] = processor(batch["sentence"]).input_ids
    return batch
```

Sau khi đã hoàn thành, ta sẽ tiến hành mapping nó trên tập train và test của dữ liệu.

3.6. Giới hạn độ dài audio

Để tối ưu tốc độ training cho mô hình ta sẽ giới hạn độ dài audio đầu vào trên dataset bằng cách cắt với độ dài tối đa cho phép trên dữ liệu âm thanh là **15 giây**.

```
# Đặt độ dài tối đa đầu vào là 15s
```

```

max_input_length_in_sec = 15.0
ds_train = ds_train.filter(lambda x: x <
                             max_input_length_in_sec *
                             processor.feature_extractor.sampling_rate,
                             input_columns=["input_length"])
ds_test = ds_test.filter(lambda x: x <
                           max_input_length_in_sec *
                           processor.feature_extractor.sampling_rate,
                           input_columns=["input_length"])

```

3.7. Tạo trình đối chiếu dữ liệu DataCollator

Dữ liệu audio đầu vào và văn bản đầu ra sẽ được padding theo cơ chế CTC và các cài đặt processor chúng ta đã chuẩn bị phía trên thông qua lớp này.

```

# Trình đối chiếu dữ liệu
data_collator =
DataCollatorCTCWithPadding(processor=processor,
                             padding=True)

```

Với các phần ta pad sẽ được điền đầy bằng cách thêm giá trị **-100** vào để bỏ qua tính mất mát (loss).

3.8. Tạo trình tính độ đo metrics

Đối với bài toán Speech-to-Text thì ngoài sai số loss ra chúng ta sẽ dùng độ đo Word error rate (WER) để ước lượng hiệu suất của mô hình.

```

# Load hàm tính toán chỉ số wer
wer_metric = load_metric("wer")

```

Lưu ý là chỉ số WER này chỉ được tính trên tập dữ liệu kiểm thử (test) để kết quả khách quan nhất có thể. Ngoài ra đối với token có giá trị **-100** tức là pad dùng để điền đầy độ dài văn bản ta đã cài đặt lúc đầu sẽ được chuyển thành pad_token_id của tokenizer.

```

# Hàm tính chỉ số wer trên tập validation
def compute_metrics(pred):
    pred_logits = pred.predictions
    pred_ids = np.argmax(pred_logits, axis=-1)

    pred.label_ids[pred.label_ids == -100] =
        processor.tokenizer.pad_token_id

```



```

pred_str = processor.batch_decode(pred_ids)
# Không nhóm các tokens (group_tokens) khi tính metrics
label_str = processor.batch_decode(pred.label_ids,
                                   group_tokens=False)

wer = wer_metric.compute(predictions=pred_str,
                        references=label_str)

return {"wer": wer}

```

3.9. Tạo mô hình huấn luyện

Ta sẽ tiến hành load mô hình từ **facebook/wav2vec2-large-xlsr-53** về rồi thiết đặt các thông số như dropout và các thông số khác như bên dưới, ta cũng cần phải chuyển mô hình sang device là **cuda** để các trọng số mô hình sẽ được lưu trên GPU, đảm bảo việc tính toán và lưu trữ tận dụng được tối đa sức mạnh của phần cứng.

```

from transformers import AutoModelForCTC
# Tải model wav2vec2 gốc
model = AutoModelForCTC.from_pretrained(
    model_wav2vec2,
    attention_dropout=0.1,
    hidden_dropout=0.1,
    feat_proj_dropout=0.0,
    mask_time_prob=0.05,
    layerdrop=0.1,
    ctc_loss_reduction="mean",
    pad_token_id=processor.tokenizer.pad_token_id,
    vocab_size=len(processor.tokenizer)
).to("cuda")

```

Sau đó ta sẽ đóng băng lớp trích xuất đặc trưng để đảm bảo lớp này không bị thay đổi các giá trị trọng số.

```

# Đóng băng lớp trích xuất đặc trưng để không training lớp này
if hasattr(model, "freeze_feature_extractor"):
    model.freeze_feature_extractor()

```

Sau đó ta sẽ tạo **TrainingArguments** để thiết đặt các siêu tham số cho mô hình này. Mô hình sẽ có các siêu tham số như sau:

Hyper-Parameter	Values
batch_size	30
epochs	7 (thực tế là 21)
learning_rate	0.0003 (3e-4)
save_steps eval_steps logging_steps	50

Bảng 3: Cài đặt siêu tham số mô hình

Ta đặt batch_size trong trường hợp này là 30, mục đích là để phân dữ liệu huấn luyện thành các lô tập hợp 30 điểm mỗi lô training, giúp tăng khả năng hội tụ của mô hình, tiết kiệm thời gian khi xử lý song song cũng như tận dụng tối đa VRAM trên GPU, nếu lớn hơn 30 sẽ bị tràn VRAM. Số epochs được đặt ở đây là 7 để tạm thời, thực tế mô hình sẽ được training qua 21 epochs do thời gian training khá lâu (khoảng 1 giờ 30 phút / 7 epochs) sẽ không thuận tiện cho việc treo máy nên chiến lược chúng ta sẽ là chia ra từng đợt để train, tổng cộng sẽ có 3 đợt mỗi đợt sẽ train 7 epochs, mỗi lần train xong sẽ lưu lại mô hình và khi nào có thời gian thuận tiện việc treo máy sẽ lấy ra train tiếp tục. Ta cũng sẽ lưu lại các thông số đo được trong quá trình này. Các step ta sẽ đặt cứ 50 bước sẽ lưu và tính toán metrics thông báo 1 lần. Còn lại sẽ để theo mặc định như sau.

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="wav2vec_vietnamese",
    group_by_length=True,
    per_device_train_batch_size=30,
    gradient_accumulation_steps=2,
    evaluation_strategy="steps",
    num_train_epochs=7,
    gradient_checkpointing=True,
    save_steps=50,
    eval_steps=50,
```

```
logging_steps=50,  
learning_rate=3e-4,  
save_total_limit=2,  
)
```

Cuối cùng ta sẽ tổng hợp lại tất cả rồi đưa vào hàm trainer rồi chạy hàm train() để huấn luyện mô hình.

```
from transformers import Trainer  
  
trainer = Trainer(  
    model=model,  
    data_collator=data_collator,  
    args=training_args,  
    compute_metrics=compute_metrics,  
    train_dataset=ds_train,  
    eval_dataset=ds_test,  
    tokenizer=processor.feature_extractor,  
)
```

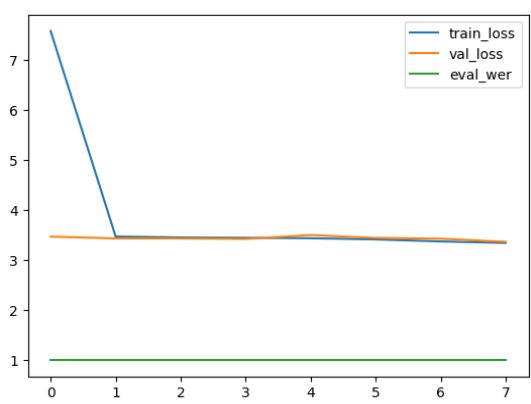
Vậy là chúng ta đã hoàn thành từ quá trình xử lý dữ liệu cho đến khởi tạo và huấn luyện mô hình, tiếp theo chúng ta sẽ tiến hành thực nghiệm để đưa ra kết quả. Toàn bộ link mã nguồn dự án sẽ được đính kèm phía cuối bài báo cáo này.

CHƯƠNG 4: THỰC NGHIỆM VÀ ĐÁNH GIÁ KẾT QUẢ

4.1. Kết quả thực nghiệm

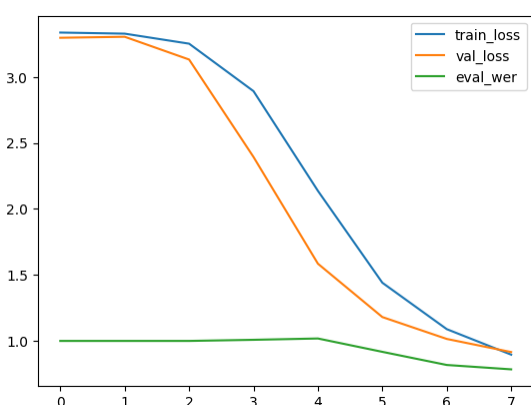
Như đã đề cập ở trên, chiến lược huấn luyện sẽ chia ra làm 3 đợt với mỗi đợt là 7 epochs vì vậy chúng ta sẽ có kết quả từng đợt như sau:

Kết quả Đợt 1:

Biểu đồ minh họa epochs [1 - 7]	Chi tiết thông số																																				
	<div><div></div><div>[441/441 1:27:23, Epoch 7/7]</div></div> <table><tr><th>Step</th><th>Training Loss</th><th>Validation Loss</th><th>Wer</th></tr><tr><td>50</td><td>7.579600</td><td>3.472332</td><td>1.000000</td></tr><tr><td>100</td><td>3.474800</td><td>3.437133</td><td>1.000000</td></tr><tr><td>150</td><td>3.456300</td><td>3.436660</td><td>1.000000</td></tr><tr><td>200</td><td>3.449500</td><td>3.426705</td><td>1.000000</td></tr><tr><td>250</td><td>3.440400</td><td>3.504475</td><td>1.000000</td></tr><tr><td>300</td><td>3.416600</td><td>3.446950</td><td>1.000000</td></tr><tr><td>350</td><td>3.376400</td><td>3.432195</td><td>1.000000</td></tr><tr><td>400</td><td>3.346800</td><td>3.368752</td><td>1.000000</td></tr></table>	Step	Training Loss	Validation Loss	Wer	50	7.579600	3.472332	1.000000	100	3.474800	3.437133	1.000000	150	3.456300	3.436660	1.000000	200	3.449500	3.426705	1.000000	250	3.440400	3.504475	1.000000	300	3.416600	3.446950	1.000000	350	3.376400	3.432195	1.000000	400	3.346800	3.368752	1.000000
Step	Training Loss	Validation Loss	Wer																																		
50	7.579600	3.472332	1.000000																																		
100	3.474800	3.437133	1.000000																																		
150	3.456300	3.436660	1.000000																																		
200	3.449500	3.426705	1.000000																																		
250	3.440400	3.504475	1.000000																																		
300	3.416600	3.446950	1.000000																																		
350	3.376400	3.432195	1.000000																																		
400	3.346800	3.368752	1.000000																																		

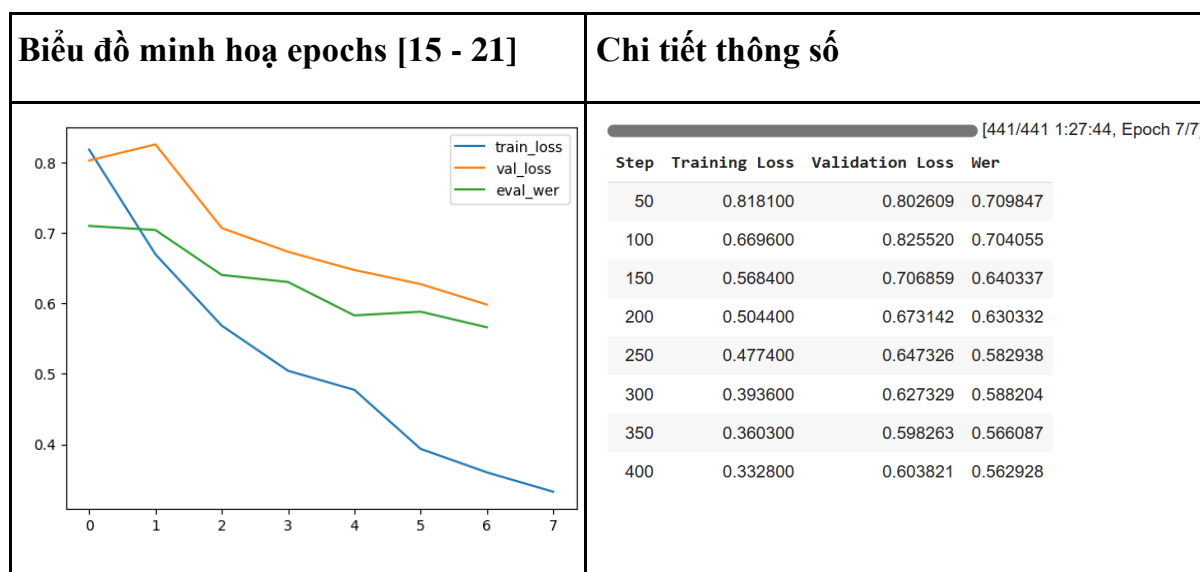
Bảng 4: Kết quả huấn luyện đợt 1 (epochs 1 - 7)

Kết quả Đợt 2:

Biểu đồ minh họa epochs [8 - 14]	Chi tiết thông số																																				
	<div><div></div> [441/441 1:28:04, Epoch 7/7]</div> <table><tr><th>Step</th><th>Training Loss</th><th>Validation Loss</th><th>Wer</th></tr><tr><td>50</td><td>3.335700</td><td>3.296169</td><td>1.000000</td></tr><tr><td>100</td><td>3.327400</td><td>3.303706</td><td>1.000000</td></tr><tr><td>150</td><td>3.251400</td><td>3.131309</td><td>1.000000</td></tr><tr><td>200</td><td>2.891700</td><td>2.393150</td><td>1.008425</td></tr><tr><td>250</td><td>2.136400</td><td>1.584817</td><td>1.018431</td></tr><tr><td>300</td><td>1.440700</td><td>1.180770</td><td>0.917325</td></tr><tr><td>350</td><td>1.089200</td><td>1.015057</td><td>0.817799</td></tr><tr><td>400</td><td>0.896000</td><td>0.915678</td><td>0.784623</td></tr></table>	Step	Training Loss	Validation Loss	Wer	50	3.335700	3.296169	1.000000	100	3.327400	3.303706	1.000000	150	3.251400	3.131309	1.000000	200	2.891700	2.393150	1.008425	250	2.136400	1.584817	1.018431	300	1.440700	1.180770	0.917325	350	1.089200	1.015057	0.817799	400	0.896000	0.915678	0.784623
Step	Training Loss	Validation Loss	Wer																																		
50	3.335700	3.296169	1.000000																																		
100	3.327400	3.303706	1.000000																																		
150	3.251400	3.131309	1.000000																																		
200	2.891700	2.393150	1.008425																																		
250	2.136400	1.584817	1.018431																																		
300	1.440700	1.180770	0.917325																																		
350	1.089200	1.015057	0.817799																																		
400	0.896000	0.915678	0.784623																																		

Bảng 5: Kết quả huấn luyện đợt 2 (epochs 8 - 14)

Kết quả Đợt 3:

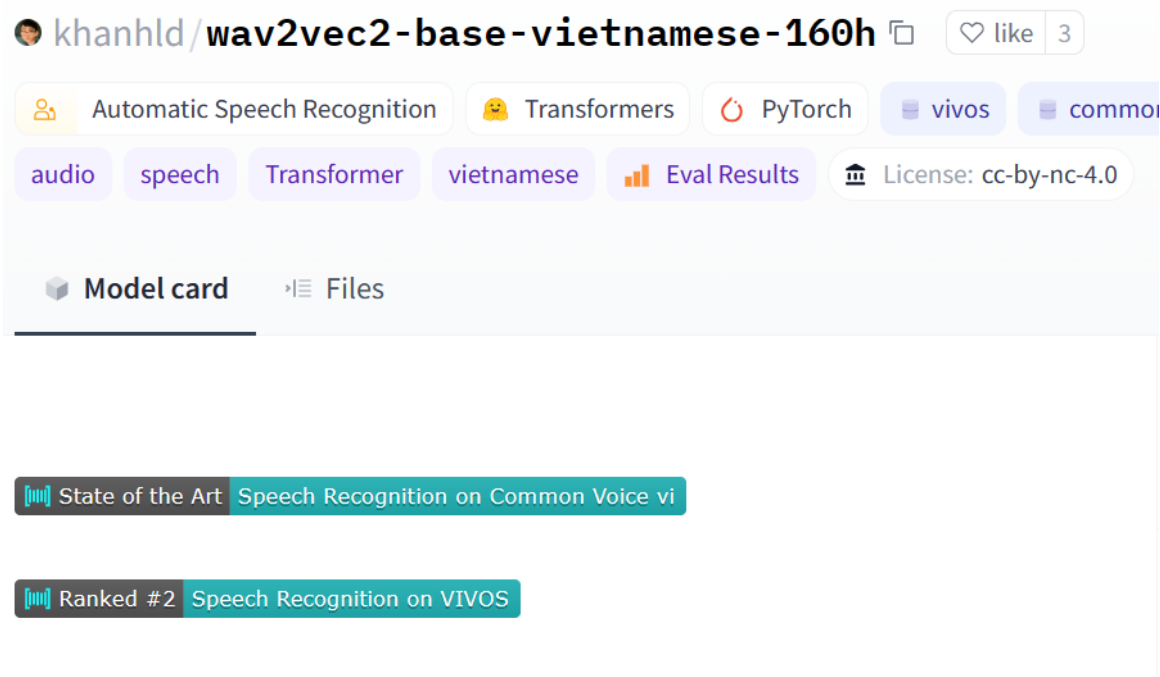


Bảng 6: Kết quả huấn luyện đợt 3 (epochs 15 - 21)

Đánh giá kết quả: Chúng ta nhận thấy rằng, ở 7 epochs đợt đầu tiên mô hình vẫn chưa có sự thay đổi nào ở chỉ số WER nhưng loss trên tập training giảm đáng kể và gần như dừng lại gần bằng loss validation. Tiếp tục ở đợt 2 sự thay đổi rõ ràng hơn ta có thể quan sát được bằng mắt khi loss trên hai tập giảm đều theo thời gian xuống dưới 1 và WER đã có sự thay đổi. Qua đợt 3 ta cũng thấy rằng loss và WER cũng tiếp tục giảm đều và chưa thấy xuất hiện hiện tượng quá khớp (overfitting) trên mô hình. Đây là một kết quả đáng mong đợi tuy nhiên để chính xác hơn ta sẽ cần training trên nhiều bộ dữ liệu khác nhau với từng vùng miền và tăng epochs để mô hình có độ chính xác cao hơn.

4.2. So sánh với các mô hình nhận diện giọng nói khác

Hiện tại, ở thời điểm đang viết bài báo cáo này, mô hình được công bố có số điểm WER tốt nhất trên Tiếng Việt là **khanhld/wav2vec2-base-vietnamese-160h**. Mô hình này đạt chỉ số WER trên tập **test** bộ Tiếng Việt của **Common Voice 8.0** là **10.78** đây chính là chỉ số cao nhất từng ghi nhận và được cho là state-of-the-art (SOTA) trên tiếng Việt hiện tại.



Hình 10: Mô hình SOTA cho Tiếng Việt trên bộ Common Voice 8.0

Ta sẽ thử lấy mô hình hiện có của chúng ta để so sánh với mô hình SOTA này, sẽ có kết quả như sau:

Model	WER
khanhld/wav2vec2-base-vietnamese-160h (SOTA)	10.78
Our Model	14.06

Bảng 7: So sánh mô hình hiện tại với mô hình SOTA

Đánh giá: Mặc dù đã đến rất gần nhưng mô hình của chúng ta vẫn chưa thể vượt qua mô hình SOTA hiện có. Nguyên nhân có thể được đưa ra là do giới hạn về phần cứng, thời gian huấn luyện cũng như độ lớn của tập dữ liệu huấn luyện còn hạn chế. Nếu được đầu tư phát triển cũng như được sự đóng góp tích cực từ cộng đồng và kết hợp với nhiều phương pháp mô hình ngôn ngữ khác nhau, mô hình này có thể đạt được kết quả vượt sự kỳ vọng.

4.3. Video demo thử nghiệm mô hình

Video demo giao diện sử dụng:

QR code	Link Youtube
	<a data-bbox="810 297 1246 338" href="https://youtu.be/Usf5wT35Qfo">https://youtu.be/Usf5wT35Qfo

CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

5.1. Kết luận

Nghiên cứu này không chỉ giải quyết một vấn đề quan trọng mà còn đóng góp vào sự phát triển của lĩnh vực và cung cấp một cơ sở cho các nghiên cứu và ứng dụng trong tương lai.

Tóm lại, trong đề án này, chúng ta đã thực hiện thành công quá trình fine tuning model **facebook/wav2vec2-large-xlsr-53** cho tiếng Việt bằng thư viện **Wav2vec2** có CTC. Kết quả độ đo WER trên tập validation là 0.562928. Kết quả này cho thấy hệ thống đã đạt được độ chính xác tương đối ở mức kỳ vọng.

Để đạt được kết quả này, ta đã thực hiện qua các bước bao gồm:

- Tiền xử lý dữ liệu âm thanh tiếng Việt trong bộ **common_voice_11**
- Tạo bộ processor, áp dụng các thay đổi lên trên bộ dữ liệu.
- Finetuning model mô hình với các siêu tham số được tối ưu hóa.
- Đánh giá độ chính xác của model trên tập validation.

Tuy nhiên, vẫn còn một số vấn đề cần được cải thiện, chẳng hạn như:

- Độ chính xác của hệ thống có thể được cải thiện bằng cách tăng kích thước tập dữ liệu huấn luyện.
- Có thể sử dụng các kỹ thuật finetuning khác để cải thiện độ chính xác của hệ thống.
- Có thể sử dụng các mô hình ngôn ngữ lớn (LLM) để cải thiện khả năng hiểu ngôn ngữ tự nhiên của hệ thống.

5.2. Hướng phát triển

Nghiên cứu có thể mở rộng để áp dụng mô hình Speech-to-Text vào các ứng dụng thực tế như tạo phụ đề cho video, triển khai trợ lý ảo hoặc tích hợp vào các sản phẩm và dịch vụ liên quan đến ngôn ngữ tiếng Việt.


Để cải thiện độ chính xác của model, chúng ta có thể thực hiện các bước sau:

- **Tăng kích thước của tập dữ liệu tiếng Việt.** Điều này có thể được thực hiện bằng cách thu thập thêm dữ liệu từ các nguồn khác nhau, chẳng hạn như các cuộc hội thoại, bản ghi âm, v.v.

- **Sử dụng các kỹ thuật tăng cường dữ liệu** để tạo ra thêm dữ liệu từ tập dữ liệu hiện có. Một số kỹ thuật tăng cường dữ liệu phổ biến bao gồm:
 - **Phản chiếu:** Lặp lại các mẫu dữ liệu hiện có theo chiều ngược lại.
 - **Trộn:** Kết hợp các mẫu dữ liệu hiện có với nhau.
 - **Thêm tiếng ồn:** Thêm tiếng ồn vào các mẫu dữ liệu hiện có.
- **Sử dụng các kỹ thuật tối ưu hóa khác nhau.** Các kỹ thuật tối ưu hóa khác nhau có thể được sử dụng để cải thiện độ chính xác của model. Một số kỹ thuật tối ưu hóa phổ biến bao gồm:
 - **AdamW:** Một biến thể của Adam sử dụng trọng số giảm động.
 - **Adagrad:** Một kỹ thuật tối ưu hóa dựa trên độ dốc.
 - **RMSProp:** Một kỹ thuật tối ưu hóa dựa trên bình phương sai số trung bình.

Ngoài ra, chúng ta cũng có thể thử nghiệm các model khác nhau để xem liệu chúng có thể đạt được độ chính xác cao hơn hay không. Hy vọng rằng những ý tưởng này sẽ giúp cải thiện độ chính xác của model và làm cho nó trở nên hữu ích hơn cho các ứng dụng thực tế.

5.3. Mã nguồn đề tài

QR code	Link Code
	https://bit.ly/3PEAOiK

TÀI LIỆU THAM KHẢO

- [1] N. T. M. Thanh, P. X. Dung, N. N. Hay, L. N. Bich, and D. X. Quy, ‘Đánh giá các hệ thống nhận dạng giọng nói tiếng việt (vais, viettel, zalo, fpt và google) trong bản tin’, *Journal of Technical Education Science*, no. 63, pp. 28–35, Apr. 2021.
- [2] T. Đ. Minh and N. T. Luận, ‘Using fuzzy logic in Vietnamese speech recognition’, in *Tạp Chí Khoa Học và Công Nghệ-Đại Học Đà Nẵng*, 2014, pp. 64–70.
- [3] C. Wang *et al.*, ‘fairseq S2T: Fast Speech-to-Text Modeling with fairseq’, *arXiv [cs.CL]*, 11-Oct-2020.
- [4] S. Bansal, H. Kamper, A. Lopez, and S. Goldwater, ‘Towards speech-to-text translation without speech recognition’, *arXiv [cs.CL]*, 13-Feb-2017.
- [5] M. Malik, M. K. Malik, K. Mehmood, and I. Makhdoom, ‘Automatic speech recognition: a survey’, *Multimed. Tools Appl.*, vol. 80, no. 6, pp. 9411–9457, Mar. 2021.
- [6] G. Saha, S. Chakroborty, and S. Senapati, ‘A new silence removal and endpoint detection algorithm for speech and speaker recognition applications’, in *Proceedings of the 11th national conference on communications (NCC)*, 2005, pp. 291–295.
- [7] B. Yegnanarayana and R. N. J. Veldhuis, ‘Extraction of vocal-tract system characteristics from speech signals’, *IEEE Trans. Speech Audio Process.*, vol. 6, no. 4, pp. 313–327, Jul. 1998.
- [8] I. Mporas, T. Ganchev, and M. Sifarakas, ‘Comparison of speech features on the speech recognition task’, *J. Comput. Sci.*, vol. 3, no. 8, pp. 608–616, Aug. 2007.
- [9] W. Alkhalidi, W. Fakhr, and N. Hamdy, ‘Automatic speech/speaker recognition in noisy environments using wavelet transform’, in *The 2002 45th Midwest Symposium on Circuits and Systems, 2002. MWSCAS-2002*, Tulsa, OK, USA, 2003.

- [10] V. V. Krishnan and P. B. Anto, 'Features of wavelet packet decomposition and discrete wavelet transform for malayalam speech recognition', *International Journal of Recent Trends in Engineering*, vol. 1, no. 2, 2009.
- [11] B. Zamani, A. Akbari, B. Nasersharif, and A. Jalalvand, 'Optimized discriminative transformations for speech features based on minimum classification error', *Pattern Recognit. Lett.*, vol. 32, no. 7, pp. 948–955, May 2011.
- [12] S. Ranjan, 'A discrete wavelet transform based approach to Hindi speech recognition', in *2010 International Conference on Signal Acquisition and Processing*, Bangalore, India, 2010.
- [13] D. Oshaughnessy, 'Automatic speech recognition: History, methods and challenges', *Pattern Recognition*, no. 10, pp. 2965–2979, 2008.
- [14] L. Rabiner and B. Juang, 'An introduction to hidden Markov models', *IEEE ASSP Mag.*, vol. 3, no. 1, pp. 4–16, 1986.
- [15] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, 'Speaker verification using adapted Gaussian mixture models', *Digit. Signal Process.*, vol. 10, no. 1–3, pp. 19–41, Jan. 2000.
- [16] K. Aida-zade, A. Xocayev, and S. Rustamov, 'Speech recognition using Support Vector Machines', in *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*, Baku, Azerbaijan, 2016.
- [17] E. S. Wahyuni, 'Arabic speech recognition using MFCC feature extraction and ANN classification', in *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, Yogyakarta, 2017.
- [18] A. Mohamed and K. N. R. Nair, 'HMM/ANN hybrid model for continuous Malayalam speech recognition', *Procedia Eng.*, vol. 30, pp. 616–622, 2012.
- [19] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, 'Listen, attend and spell: A neural network for large vocabulary conversational speech recognition', in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Shanghai, 2016.

- [20] G. Hemakumar and P. Punitha, ‘Speech recognition technology: a survey on Indian languages’, *International Journal of Information Science and Intelligent System*, vol. 2, no. 4, pp. 1–38, 2013.
- [21] J. W. Forgie and C. D. Forgie, ‘Results obtained from a vowel recognition computer program’, *J. Acoust. Soc. Am.*, vol. 31, no. 6_Supplement, pp. 844–844, Jun. 1959.
- [22] V. M. Velichko and N. G. Zagoruyko, ‘Automatic recognition of 200 words’, *Int. J. Man. Mach. Stud.*, vol. 2, no. 3, pp. 223–234, Jul. 1970.
- [23] H. Sakoe and S. Chiba, ‘Dynamic programming algorithm optimization for spoken word recognition’, in *Readings in Speech Recognition*, Elsevier, 1990, pp. 159–165.
- [24] F. Weninger *et al.*, ‘Speech enhancement with LSTM recurrent neural networks and its application to noise-robust ASR’, in *Latent Variable Analysis and Signal Separation*, Cham: Springer International Publishing, 2015, pp. 91–99.
- [25] G. Hinton *et al.*, ‘Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups’, *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
- [26] L. Dong, S. Xu, and B. Xu, ‘Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition’, in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, AB, 2018.
- [27] H. M. Tan, K.-W. Liang, Y.-S. Lee, C.-T. Li, Y.-H. Li, and J.-C. Wang, ‘Speech separation using augmented-discrimination learning on squash-norm embedding vector and node encoder’, *IEEE Access*, vol. 10, pp. 102048–102063, 2022.
- [28] Yu, Dong, and Lin Deng ‘Automatic speech recognition’, *Berlin: Springer*, Vol. 1, 2016.
- [29] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, ‘Front-end factor analysis for speaker verification’, *IEEE Trans. Audio Speech Lang. Processing*, vol. 19, no. 4, pp. 788–798, May 2011.

- [30] D. A. Reynolds and R. C. Rose, ‘Robust text-independent speaker identification using Gaussian mixture speaker models’, *IEEE Trans. Speech Audio Process.*, vol. 3, no. 1, pp. 72–83, 1995.
- [31] M. Fujimoto and Y. A. Riki, ‘Robust speech recognition in additive and channel noise environments using GMM and EM algorithm’, in *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, Montreal, Que., Canada, 2004.
- [32] Moon, Todd K. "The expectation-maximization algorithm.", *IEEE Signal processing magazine*, pp. 47-60, 1996.
- [33] Jelinek, F. ‘Continuous speech recognition by statistical methods’, *Proceedings of the IEEE*, 64(4), pp. 532-556, 1976.
- [34] L. Liporace, ‘Maximum likelihood estimation for multivariate observations of Markov sources’, *IEEE Trans. Inf. Theory*, vol. 28, no. 5, pp. 729–734, Sep. 1982.
- [35] L. R. Bahl, F. Jelinek, and R. L. Mercer, ‘A maximum likelihood approach to continuous speech recognition’, *IEEE transactions on pattern analysis and machine intelligence*, pp. 179-190, 1983.
- [36] P. Smolensky, *Information processing in dynamical systems: Foundations of harmony theory*. 1986.
- [37] F. Seide, G. Li, X. Chen, and D. Yu, ‘Feature engineering in Context-Dependent Deep Neural Networks for conversational speech transcription’, in *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, Waikoloa, HI, USA, 2011.
- [38] F. Seide, G. Li, and D. Yu, ‘Conversational speech transcription using context-dependent deep neural networks’, in *Twelfth annual conference of the international speech communication association*, 2011.
- [39] K. Zinchenko, C.-Y. Wu, and K.-T. Song, ‘A study on speech recognition control for a surgical robot’, *IEEE Trans. Industr. Inform.*, vol. 13, no. 2, pp. 607–615, Apr. 2017.

- [40] Baevski, Alexei, et al, ‘wav2vec 2.0: A framework for self-supervised learning of speech representations.’, *Advances in neural information processing systems* 33, 2020.
- [41] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, ‘BERT: Pre-training of deep bidirectional Transformers for language understanding’, *arXiv*, 10-Oct-2018.
- [42] D. Hendrycks and K. Gimpel, ‘Gaussian Error Linear Units (GELUs)’, *arXiv [cs.LG]*, 27-Jun-2016.
- [43] Andrews, L. C. ‘Special functions of mathematics for engineers’, *Spie Press*, Vol. 49, pp. 110, 1998.
- [44] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, ‘Roberta: A robustly optimized bert pretraining approach’, *arXiv preprint arXiv:1907.11692*, 2019.
- [45] R. Ardila *et al.*, ‘Common Voice: A massively-multilingual speech corpus’, *arXiv [cs.CL]*, 13-Dec-2019.