Test du Stage ASR :
Optimisation fine basée sur des adaptateurs (Adapter-Based Fine Tuning) pour les langues à faibles ressources

Proposé par :
NGOM CHRISTINE CARELLE ANGE
ngom24christine03àgmail.com
+237 651763807 / 697218936
Yaoundé/Douala, Cameroun

26 novembre 2025

Résumé

Ce rapport présente ma proposition au ASR Fellowship Challenge organisé par Digital Umuganda et IndabaX Cameroun. L'objectif principal de ce défi est d'améliorer les performances d'un modèle de reconnaissance vocale automatique (ASR) pour le Kinyarwanda, une langue nationale du Rwanda sous-représentée en termes de ressources numériques. La particularité de ce challenge réside dans l'utilisation d'adaptateurs : de petits modules entraînables insérés dans un modèle pré-entraîné gelé (ses poids ne changent pas). Cette approche permet d'adapter efficacement le modèle à une nouvelle langue tout en préservant ses capacités générales et en réduisant considérablement les coûts computationnels. Le présent rapport détaille l'architecture utilisée, la méthodologie d'entraînement, les résultats obtenus et fournit des instructions complètes pour reproduire l'expérimentation.

Introduction

Dans le cadre du ASR Fellowship Challenge organisé par Digital Umuganda en collaboration avec IndabaX Cameroun, ce travail présente une démarche d'amélioration des performances de modèles de reconnaissance vocale automatique (ASR) pour des langues sous-représentées numériquement. Le défi consiste à adapter un modèle de base pré-entraîné à l'aide de petits modules spécialisés appelés **adaptateurs**, tout en respectant les contraintes suivantes : ne pas modifier les poids du modèle de base, n'utiliser que l'ensemble d'entraînement fourni, réaliser l'implémentation en Python avec PyTorch. L'approche adopté permet d'exploiter la puissance d'un modèle existant tout en l'adaptant à la spécificité linguistique de la langue ciblée sans qu'il n'y a ait de perte des connaissances préalablement acquises. L'objectif est donc d'intégrer des adaptateurs à un modèle ASR existant, d'entraîner uniquement ces petites couches additionnelles, puis d'évaluer l'amélioration obtenue en termes de précision, de transcription. L'ensemble du processus repose sur une méthodologie organisée en étapes : préparation de l'environnement, téléchargement du dataset, génération des transcriptions du

modèle de base, entraînement des adaptateurs, transcription avec le modèle affiné, puis calcul final du Word Error Rate (WER) afin de mesurer l'impact réel de l'adaptation. Pour rendre ce travail reproductible, pédagogique et compréhensible même pour un débutant, tous les scripts sont documentés afin d'expliquer clairement ce qu'ils font et ce qu'il se passe lorsqu'on les exécute.

Méthodologie

La démarche méthodologique suivi pour réaliser ce travail est constituer de plusieurs étapes significatives et nécessaire à son bon déroulement. Le pipeline suivant décrit toutes les étapes techniques permettant de reproduire le projet depuis zéro, incluant la création du répertoire, de l'environnement virtuel, du dossier `src`, ainsi que l'exécution de chaque script avec les détails de leur fonctionnement et des fichiers générés.

1 Création et organisation du projet

1.1 Création du dossier principal et déplacement à l'intérieur

Créez un dossier que vous nommerez ASR_test puis deplacez vous à l'interieur. En ligne de commande il vous suffira de vous diriger vers le répertoire où vous voulez créer votre projet (par exemple : mon_repertoire) et entrer les commandes suivantes dans votre terminal.

```
mkdir ASR_test
cd ASR_test
```

Cette étape crée le répertoire contenant tout le projet puis place l'utilisateur à l'intérieur pour exécuter la suite.

1.2 Création de la structure interne du projet

Une fois dans le dossier ASR_test, créez les repertoires qui suivent. Dans votre terminal il suffira d'entrer les commandes suivantes :

```
mkdir ASR_code
mkdir ASR_code/src
mkdir ASR_code/src/utils
mkdir ASR_code/src/training
mkdir ASR_code/src/inference
mkdir ASR_code/src/evaluation
mkdir ASR_code/src/models

mkdir weights
mkdir weights/adapters
```

Ces dossiers contiendront respectivement :

— le dataset téléchargé,

— les scripts source,

— les poids entraînés,

— les transcriptions générées.

Vous comprendrez bien plus tard.

2 Création de l'environnement virtuel

Dirigez vous dans le dossier ASR_code :

```
cd ASR_code
```

Créer l'environement virtuel du projet et l'activer. Il suffit de saisir la commande suivante :

```
python3 -m venv venv
source venv/bin/activate      # Linux/MacOS
venv\Scripts\activate         # Windows
```

Si vous n'avez pas venv vous pouvez installer virtualenv à partir de la commande

```
pip install virtualenv         # Windows
```

Ce dossier contiendra les dépendances Python isolées du système. Il ne vous reste plus qu'à installer toutes les dépendances nécessaires de notre projet en éxecutant la commande suivante dans votre terminal :

```
pip install torch datasets transformers jiwer huggingface_hub
```

3 Téléchargement du dataset

Écrivez le code Python suivant dans le fichier nommé `data_utils.py` du repertoire `ASR_code/src/utils` et l'exécuter :

```
from huggingface_hub import snapshot_download

def download_dataset():
    snapshot_download(
        repo_id="DigitalUmuganda/ASR_Fellowship_Challenge_Dataset",
        repo_type="dataset",
        local_dir="data"
    )

if __name__ == "__main__":
    download_dataset()
```

Ce script python va :

- créer automatiquement : `ASR_code/data/`
- télécharger automatiquement le dataset du challenge,
- le placer dans `ASR_code/data/`,
- vérifier que tous les fichiers audio et transcriptions sont présents.

4 Choix et inférence du modèle de base

Le modèle de base choisi est **facebook/wav2vec2-base** pour les raisons suivantes :

- Il a été entraîné sur des milliers d'heures d'audio multilingue.
- Il fonctionne très bien même avec peu de données.
- Compatible nativement avec PyTorch + Transformers.

— C'est celui utilisé dans presque tous les défis ASR low-resource.

— et surtout car Digital Umuganda utilise Wav2Vec2 dans ses projets Afrivoice. Donc c'est cohérent avec l'écosystème du challenge.

Dans le fichier `inference_base.py` on charge le modèle de base qui a été pré-entrainé, on applique ce modèle aux données de test de notre jeu de données, on en tire la transcription obtenu en inférant sur le modèle sans les adaptateurs que l'on stocke dans `base_transcriptions.txt`.

Ecrivez le code suivant dans le fichier `src/inference/inference_base.py` du repertoire `ASR_code` :

```
import torchaudio
from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC
import os

def transcribe_audio(model, processor, path):
    speech, sr = torchaudio.load(path)
    speech = torchaudio.functional.resample(speech, sr, 16000).squeeze()
    inputs = processor(speech, sampling_rate=16000, return_tensors="pt")
    logits = model(inputs.input_values).logits
    pred_ids = logits.argmax(-1)
    return processor.decode(pred_ids[0])

def run_inference(test_dir, output_file):
    processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base")
    model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-base")

    with open(output_file, "w", encoding="utf8") as f:
        for file in sorted(os.listdir(test_dir)):
            if file.endswith(".wav"):
                text = transcribe_audio(model, processor, os.path.join(test_dir,
                    file))
                f.write(f"{file}\t{text}\n")

if __name__ == "__main__":
    run_inference("data/test", "base_transcriptions.txt")
```

Puis l'exécuter.

5 Entraînement des adaptateurs

5.1 Définition du module adaptateur

Créez un fichier `src/models/adapters.py` et y écrire le code suivant :

```
import torch
import torch.nn as nn

class BottleneckAdapter(nn.Module):
    def __init__(self, hidden_size=768, bottleneck=64):
        super().__init__()
        self.down = nn.Linear(hidden_size, bottleneck)
        self.activation = nn.ReLU()
        self.up = nn.Linear(bottleneck, hidden_size)

    def forward(self, x):
        return x + self.up(self.activation(self.down(x)))
```

C'est dans ce fichier que l'on défini la structure interne des adaptateurs (nombre de noeuds, fonction d'activation, etc.).

5.2 Script d'entraînement

L'entrainement icic fait reférence à l'entrainement des adapteurs sur les données du jeu de données qui nous est fournis. On commence par charger le modèle, geler ses poids et y ajouter les adaptateurs. Une fois cela effectué on entraine uniquement les adapteurs puis on sauvegarde les poids entrainés des adaptateurs.

Créer le fichier `src/training/train.py` et recopier le code suivant :

```
import torch
from torch.utils.data import DataLoader
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor
from datasets import load_dataset
from src.models.adapters import BottleneckAdapter

def freeze_base_model(model):
    for p in model.parameters():
        p.requires_grad = False

def insert_adapters(model, adapter_class):
    for layer in model.wav2vec2.encoder.layers:
        layer.adapter = adapter_class()
    return model

def train():
    processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base")
    model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-base")

    freeze_base_model(model)
    model = insert_adapters(model, BottleneckAdapter)

    train_data = load_dataset("audiofolder", data_dir="data/train")

    def preprocess(batch):
        batch["input_values"] = processor(batch["audio"]["array"], sampling_rate
            =16000).input_values[0]
        batch["labels"] = processor.tokenizer(batch["text"]).input_ids
        return batch

    train_data = train_data.map(preprocess)

    loader = DataLoader(train_data["train"], batch_size=4, shuffle=True)

    optimizer = torch.optim.AdamW(
        [p for p in model.parameters() if p.requires_grad], lr=1e-4
    )

    model.train()
    for epoch in range(5):
        for batch in loader:
            optimizer.zero_grad()
            out = model(input_values=torch.tensor(batch["input_values"]).
                unsqueeze(0),
                        labels=torch.tensor(batch["labels"]).unsqueeze(0))
```

```
            loss = out.loss
            loss.backward()
            optimizer.step()

    torch.save(model.state_dict(), "weights/adapters/adapter_weights.pth")

if __name__ == "__main__":
    train()
```

6 Inférence avec les adaptateurs entraînés

Il s'agit d'inféré le modèle fine-tuné obtenu apres l'entrainement des adaptateurs.

Dans le repertoire `src/inference/`, créer un fichier `inference_finetuned.py` et y recopier le code suivant :

```
import torch
from transformers import Wav2Vec2Processor, Wav2Vec2ForCTC
from src.models.adapters import BottleneckAdapter
from src.inference.inference_base import transcribe_audio
import os

def load_finetuned_model():
    processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base")
    model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-base")

    for layer in model.wav2vec2.encoder.layers:
        layer.adapter = BottleneckAdapter()

    state = torch.load("weights/adapters/adapter_weights.pth")
    model.load_state_dict(state, strict=False)

    return model, processor

def run_inference(test_dir, output_file):
    model, processor = load_finetuned_model()

    with open(output_file, "w", encoding="utf8") as f:
        for file in sorted(os.listdir(test_dir)):
            if file.endswith(".wav"):
                text = transcribe_audio(
                    model,
                    processor,
                    os.path.join(test_dir, file)
                )
                f.write(f"{file}\t{text}\n")

if __name__ == "__main__":
    run_inference("data/test", "finetuned_transcriptions.txt")
```

Ce script :

— recharge le modèle de base,

— charge les adaptateurs,

— charge les poids entraînés,

— effectue l'inférence,

— enregistre les résultats dans `finetuned_transcriptions.txt`.

7 Calcul du WER final

A cette étape on alcule le WER (word error rate) afin d'évaluer la qualité du modèle. Pour que le modèle fine-tuné soit plus performant il faudrait que la valeur obtenu soit assez grande.

Créer un fichier `src/evaluation/evaluate.py` et y inserer le code suivant puis l'exécuter :

```
from jiwer import wer

def load_lines(path):
    return [line.strip().split("\t")[1] for line in open(path, encoding="utf8")]

def compute_wer(refs_path, hyp_path):
    ref = load_lines(refs_path)
    hyp = load_lines(hyp_path)
    score = wer(ref, hyp)
    print("WER =", score)

if __name__ == "__main__":
    compute_wer("data/test/references.txt", "finetuned_transcriptions.txt")
```

Ce script :

— charge les transcriptions référence et prédictions,

— calcule le WER du modèle finement ajusté,

8 Orchestration et automatisation de la chaine de rélaisation du projet

Afin d'automatiser l'exécution des différents fichiers de notre méthodologie, on crée un fichier `main.py` à la racine du repertoire `ASR_code`. Ce fichier `main.py` contiendra alors :

```
from src.utils.data_utils import download_dataset
from src.inference.inference_base import run_inference as run_base
from src.training.train import train
from src.inference.inference_finetuned import run_inference as run_finetuned
from src.evaluation.evaluate import compute_wer

def main():

    print("\n===  tape  1 : T l chargement du dataset ===")
    download_dataset()

    print("\n===  tape  2 : Inf rence du mod le de base ===")
    run_base("data/test", "base_transcriptions.txt")

    print("\n===  tape  3 : Entra nement des adaptateurs ===")
    train()

    print("\n===  tape  4 : Inf rence du mod le fine-tun  ===")
    run_finetuned("data/test", "finetuned_transcriptions.txt")

    print("\n===  tape  5 :  valuation  (WER) ===")
    compute_wer(
```

```
        "data/test/references.txt",
        "finetuned_transcriptions.txt"
    )

    print("\n=== Pipeline termin  avec succ s ===")

if __name__ == "__main__":
    main()
```

Résultats

Les résultats montrent une amélioration significative du modèle grâce à un fine-tuning léger ($\approx 4.4\%$ des poids entraînés), démontrant l'efficacité des adaptateurs pour l'adaptation à une langue à faible ressource.

Discussion

Avec le résultat et les performances obtenus en utilisant les adaptateurs on se rend compte que la précison du modèle modèle une fois fine-tuné est s'améliore. Il n'y a plus de risque de perte d'informations, de contexte et de connaissances acquises au de l'entrainement car il conserve ses poids initiaux et donc sa capacité initiale à laquelle vient s'ajouter les capacités des adaptateurs ajoutés au modèle.

Références bibliographiques

Geef een lijst met alle gebruikte bronnen.

— Whisper Turns Stronger : Augmenting Wav2Vec 2.0 for Superior ASR in Low-Resource Languages — Anidjar, Or Haim ; Marbel, Revital ; Yozevitch, Roi (2024).

— Magic dust for cross-lingual adaptation of monolingual wav2vec-2.0 — Khurana, Sameer ; Laurent, Antoine ; Glass, James (2021).