Multilingual Speech Recognition and Translation using Wav2Vec2 and Whisper

Subtitle: An end-to-end guide to detect spoken language, transcribe, translate, and synthesize speech using pretrained transformers and a custom neural classifier.

Abstract

This report provides a step-by-step, end-to-end description of a project that: (1) extracts speech embeddings using the pretrained Wav2Vec2 model, (2) trains a small neural network classifier to perform language identification, (3) transcribes audio using OpenAI Whisper, (4) translates text using Google Translate (or an offline alternative), and (5) synthesizes spoken translations with gTTS (or an offline TTS). The report includes design rationale, dataset structure, code explanations, Colab-ready commands, inference & GUI instructions, troubleshooting tips, and extensions.

Table of Contents

---

1. Project Overview

Goal: Build a pipeline that automatically detects which language is spoken in an audio clip, transcribes the speech to text, translates that text to a target language, and speaks the translated text.

High-level pipeline:

Audio (.wav/.mp3/.mp4) -> Wav2Vec2 feature extractor -> 2048-d embedding (mean+max pooling) -> NN classifier -> language label

Then: audio -> Whisper -> transcription -> translate -> gTTS -> audio playback

Why this approach? Pretrained models (Wav2Vec2 and Whisper) provide powerful multilingual feature extraction & ASR without training huge models from scratch. A small classifier on top of fixed Wav2Vec2 embeddings is efficient and quick to train for language identification.

---

2. Requirements & Setup

Hardware

- GPU recommended (Colab GPU or local CUDA-enabled GPU) for faster embedding extraction and Whisper inference.
- CPU-only is possible but slower.

Python packages

Recommended pip command (Colab-friendly):

pip install torch torchvision torchaudio transformers librosa soundfile joblib tqdm moviepy googletrans==4.0.0-rc1 gTTS ipywidgets openai-whisper pydub

Note: googletrans and gTTS require internet. Replace with offline alternatives if needed (instructions later).

Files & Paths

- DATASET_DIR — directory with subfolders per language, each containing .wav files.
- MODEL_PATH, SCALER_PATH, LABELS_PATH — where to save your trained artifacts (e.g., Google Drive paths in Colab).

---

3. Dataset Organization

Organize your data as:

augmented_dataset/

english_augmented/

file1.wav

file2.wav

hindi_augmented/

file1.wav

tamil_augmented/

...

Each folder corresponds to a class label. Keep audio at decent quality (16 kHz recommended) and a balanced number of samples per language if possible.

Tips:

- For small datasets, use augmentation (noise, pitch, time-stretch) to increase robustness.
- Use consistent sample rate (16 kHz) for Wav2Vec2.

---

4. Feature Extraction

Model used: facebook/wav2vec2-large-xlsr-53 (Hugging Face)

Method:

1. Load audio at 16 kHz with librosa.load(..., sr=16000, mono=True).
2. Pass raw waveform through Wav2Vec2 feature extractor and model.
3. Obtain last_hidden_state tensor shape (1, seq_len, 1024).
4. Pool across time via mean and max pooling: torch.cat([mean(dim=1), max(dim=1).values], dim=1) → 2048-d vector.

Why mean+max? Mean captures average behavior; max retains salient peaks. Combining yields a richer, fixed-length representation.

Example function (concept):

returns numpy 2048-d vector

def extract_embedding(file_path, augment=False):

speech, sr = librosa.load(file_path, sr=16000, mono=True)

inputs = feature_extractor(speech, sampling_rate=16000, return_tensors='pt', padding=True)

```
    with torch.no_grad():
        hidden_states = base_model(**inputs).last_hidden_state
        embedding = torch.cat([hidden_states.mean(dim=1),
hidden_states.max(dim=1).values], dim=1)
        return embedding.squeeze().cpu().numpy()
```

---

5. Classifier Architecture & Training

Neural network (used in project)

- Input: 2048
- Dense 512 → BatchNorm → ReLU → Dropout(0.2)
- Dense 256 → BatchNorm → ReLU → Dropout(0.2)
- Output: num_classes (linear), CrossEntropyLoss

Why a custom NN?

- Wav2Vec2 provides universal speech features. A lightweight classifier learns language-specific patterns in those features without retraining the huge backbone.

Training tips

- Scale embeddings with StandardScaler() (fit on training embeddings).
- Use train_test_split(..., stratify=y) to preserve class distribution.
- Use Adam optimizer (lr ~ 5e-4), batch size 32, epochs 25–50 depending on dataset size.
- Save model.state_dict(), scaler, and labels with joblib/torch.save.

Training pseudo-loop:

```
for epoch in range(epochs):
    model.train()
    for xb,yb in batches:
        optimizer.zero_grad(); out=model(xb); loss=crit(out,yb)
        loss.backward(); optimizer.step()
```

---

6. Evaluation

- After training, evaluate on held-out test set.
- Use classification_report (precision, recall, f1) and accuracy_score.
- Optionally compute confusion matrix.

Important: If some classes have few samples, prefer macro-F1 over accuracy.

---

7. Inference Pipeline (Audio / Video / Mic)

1. Audio/Video input → if video, extract audio with moviepy at 16kHz.
2. Language prediction: extract embedding → scale → pass through classifier → language label.
3. Transcription: use whisper to transcribe audio to text.
4. Translation: use googletrans to translate into target language (or offline alternative).
5. TTS: use gTTS to synthesize the translated text (or offline alternative).

Edge cases:

- Very short utterances might produce poor embeddings — set a minimum length or pad.
- Noisy audio reduces accuracy — consider pre-processing (denoising, VAD).

---

8. Colab Notebook — Step-by-step (commands to run)

1. Mount drive:

```
from google.colab import drive

drive.mount('/content/drive')
```

2. Install dependencies (run in a cell):

```
!pip install torch torchvision torchaudio --extra-index-url
https://download.pytorch.org/whl/cu118

!pip install transformers librosa soundfile joblib tqdm moviepy googletrans==4.0.0-rc1
gTTS ipywidgets openai-whisper pydub
```

3. Upload dataset to drive/MyDrive/augmented_dataset or point DATASET_DIR there.
4. Run training script cell (the training code you have). Monitor GPU RAM & time.

5. Save artifacts to Drive:

torch.save(model.state_dict(), '/content/drive/MyDrive/language_classifier_nn.pth')

joblib.dump(scaler, '/content/drive/MyDrive/language_scaler.pkl')

joblib.dump(labels, '/content/drive/MyDrive/language_labels.pkl')

6. Run the inference GUI cells. If using mic recording, ensure browser permissions enabled.

---

9. GUI and Interaction

- Uses ipywidgets to create a simple interface with: file upload (audio/video), mic recording button (JS), and a translate-to dropdown.
- output_area = widgets.Output() captures prints for cleaner UI.
- The JS snippet records audio in the browser, sends Base64 to Python via google.colab.kernel.invokeFunction.

Important Colab note: JS mic recording will only work in browsers that allow microphone access and inside the Colab environment.

---

10. Deployment & Offline Options

If you want a fully offline pipeline:

- **Translation:** replace googletrans with Argos Translate (offline models available).
- pip install argostranslate
- # download language packs and use argos API
- **TTS:** replace gTTS with pyttsx3 (purely local) or Coqui TTS for higher-quality models.
- pip install pyttsx3

Replace gTTS example:

import pyttsx3

engine = pyttsx3.init()

engine.save_to_file(text, 'output.wav')

engine.runAndWait()

Argos example (simple):

```
import argostranslate.package, argostranslate.translate

# install packages and call translate.translate(text, from_code, to_code)
```

---

11. Diagrams (Graphviz)

Pipeline diagram (Graphviz dot):

```
digraph pipeline {
  rankdir=LR;
  node [shape=box, style=rounded, color=black];
  Audio [label="Input Audio\n(.wav/.mp3/.mp4)"];
  W2V [label="Wav2Vec2\nFeature Extractor\n(last_hidden_state)"];
  Pool [label="Mean + Max Pooling\n-> 2048-d"];
  NN [label="Language Classifier\n(512->256->C)"];
  Whisper [label="Whisper\n(Transcription)"];
  Translate [label="Translate\n(googletrans/Argos)"];
  TTS [label="TTS\n(gTTS/pyttsx3)"];

  Audio -> W2V -> Pool -> NN -> LangLabel;
  Audio -> Whisper -> Transcript;
  Transcript -> Translate -> TranslatedText -> TTS;
}
```

You can render this dot graph using Graphviz or online viewers.

---

12. Troubleshooting & Tips

- **Out of Memory**: If Wav2Vec2 or Whisper causes CUDA OOM, reduce batch sizes, use smaller Whisper model (medium/small), or move to CPU for Whisper.
- **Slow inference**: Whisper large is slow — medium or small may be fast enough.
- **Bad language predictions**: Ensure labels and scaler were saved/loaded correctly. Confirm that pooling and embedding size match training (2048). If you changed pooling steps, classifier input_dim must match.

- **No audio playback**: In Colab, use IPython.display.Audio or download the file and play locally.
- **gTTS language codes**: Ensure your lang_code_map matches gTTS supported codes.

---

13. Future Work & Extensions

- Fine-tune Wav2Vec2 (instead of frozen features) with a classification head for potentially higher accuracy.
- Use pretrained language ID models (e.g., pycld3 for text or langid for quick checks) as baselines.
- Replace Whisper with a lighter ASR for real-time applications.
- Build a web app (FastAPI/Streamlit) for production usage.
- Add language confidence scores and reject low-confidence predictions.

---

14. References

- Wav2Vec 2.0: https://arxiv.org/abs/2006.11477
- Hugging Face Wav2Vec2: https://huggingface.co/facebook/wav2vec2-large-xlsr-53
- OpenAI Whisper: https://github.com/openai/whisper
- Argos Translate: https://github.com/argosopentech/argos-translate
- gTTS: https://pypi.org/project/gTTS/

Appendix: Useful Code Snippets

Save & Load artifacts

```
# Save
torch.save(model.state_dict(), '/content/drive/MyDrive/language_classifier_nn.pth')
joblib.dump(scaler, '/content/drive/MyDrive/language_scaler.pkl')
joblib.dump(labels, '/content/drive/MyDrive/language_labels.pkl')


# Load
```

```
scaler = joblib.load('/content/drive/MyDrive/language_scaler.pkl')

labels = joblib.load('/content/drive/MyDrive/language_labels.pkl')

model.load_state_dict(torch.load('/content/drive/MyDrive/language_classifier_nn.pth'
, map_location=device))
```

Inference example (single file)

```
emb = extract_embedding('example.wav')

emb_scaled = scaler.transform([emb])

tensor = torch.tensor(emb_scaled, dtype=torch.float32).to(device)

with torch.no_grad():

    out = model(tensor)

    pred_idx = torch.argmax(out, dim=1).item()

print('Predicted:', labels[pred_idx])
```
