Task 4

Compare your finetuned model from task 3 to be compared against the results from task 2a for the cv-valid-dev mp3 dataset. Describe your observations and propose a series of steps (including datasets and experiments) to improve the accuracy. Your answer can be saved as training-report.pdf under the main repository

Please find the answer at the end of this document.

In [1]:
```
import os
import pandas as pd
import torch
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor
import librosa
import numpy as np
import evaluate
from tqdm import tqdm
import matplotlib.pyplot as plt
```
In [2]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```
In [3]:
```
# Load the model and tokenizer
model_name = "facebook/wav2vec2-base-960h"
model = Wav2Vec2ForCTC.from_pretrained(model_name).to("cuda" if torch.cuda.is_available() else "cpu")
processor = Wav2Vec2Processor.from_pretrained(model_name)
```
Some weights of Wav2Vec2ForCTC were not initialized from the model checkpoint at facebook/wav2vec2-base-960h and are newly initialized: ['wav2vec2.masked_sp
ec_embed']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

In [4]:
```
# Path to the folder that contains the mp3 files and the CSV file for cv-valid-dev
cv_folder = 'C:/Users/lingy/OneDrive/Desktop/HTX/HTX_techtest/common_voice/'  # Update this to your local path
cv_valid_dev_audio_folder = os.path.join(cv_folder, 'cv-valid-dev/')
cv_valid_dev_csv_file = os.path.join(cv_folder, 'cv-valid-dev.csv')

# Load the CSV file into a dataframe
cv_valid_dev_df = pd.read_csv(cv_valid_dev_csv_file)

# Update the 'filename' column to include the full path to the audio files
cv_valid_dev_df['filename'] = cv_valid_dev_df['filename'].apply(lambda x: os.path.join(cv_valid_dev_audio_folder, x))
```
In [5]:
```
def preprocess_audio(file_path, target_sr=16000):
    # Load audio file using librosa
    audio, sr = librosa.load(file_path, sr=None)

    # Resample audio if needed
    if sr != target_sr:
        audio = librosa.resample(audio, orig_sr=sr, target_sr=target_sr)

    # Normalize the audio to [-1, 1]
    audio = audio / np.max(np.abs(audio))

    # Convert to tensor for Wav2Vec2 processor
    input_values = processor(audio, sampling_rate=target_sr, return_tensors="pt").input_values
    input_values = input_values.squeeze(0)
    input_values = input_values.unsqueeze(0)
    return input_values  # Remove batch dimension
```
In [6]:

```python
def transcribe_audio(file_path):
    # Preprocess the audio file
    input_values = preprocess_audio(file_path)

    # Move input values to the same device as the model
    input_values = input_values.to(device)

    # Perform inference (model outputs logits)
    with torch.no_grad():
        logits = model(input_values).logits

    # Use argmax to get the most probable token ids
    predicted_ids = torch.argmax(logits, dim=-1)

    # Decode the token ids to text
    transcription = processor.batch_decode(predicted_ids)

    return transcription[0]
```

In [ ]:

```python
# Create a new column with transcriptions
cv_valid_dev_df['generated_text_fine-tuned'] = cv_valid_dev_df['filename'].apply(lambda x: transcribe_audio(x))

# Save the updated dataframe to a new CSV file
cv_valid_dev_df['generated_text_fine-tuned'] = cv_valid_dev_df['generated_text_fine-tuned'].str.lower()
cv_valid_dev_df.to_csv(cv_valid_dev_csv_file, index=False)
```

In [11]:

```python
# Initialize the WER metric from the evaluate library
wer_metric = evaluate.load("wer")
```

In [12]:

```python
# Define a function to compute WER for the text generated from "fine-tuned" model with the groundtruth for cv_valid_dev dataset
def evaluate_model_generated_text_fine_tuned(df):
    predictions = []
    references = []

    for idx, row in tqdm(df.iterrows(), total=len(df)):
        # Get the file path and ground truth transcription
        predicted_text = row['generated_text_fine-tuned']
        ground_truth = row['text']

        # Append predictions and references for evaluation
        predictions.append(predicted_text)
        references.append(ground_truth)

    # Compute the Word Error Rate (WER)
    wer_score = wer_metric.compute(predictions=predictions, references=references)
    return wer_score
```

In [13]:

```python
# Evaluate the text generated from "fine-tuned" model with the groundtruth for cv_valid_dev dataset
wer_score = evaluate_model_generated_text_fine_tuned(cv_valid_dev_df)

# Log the overall performance
print(f"Word Error Rate (WER) on the cv_valid_dev set (groundtruth VS fine-tuned): {wer_score * 100:.2f}%")
```

```
100%|██████████| 4076/4076 [00:00<00:00, 20875.03it/s]
Word Error Rate (WER) on the cv_valid_dev set (groundtruth VS fine-tuned): 14.39%
```

In [19]:

```python
# Define a function to compute WER for the generated_text output from Task 2 with the groundtruth for cv_valid_dev dataset
def evaluate_model_generated_text_task2(df):
    predictions = []
    references = []

    for idx, row in tqdm(df.iterrows(), total=len(df)):
        # Get the file path and ground truth transcription
        predicted_text = row['generated_text']
        ground_truth = row['text']

        # Append predictions and references for evaluation
        predictions.append(predicted_text)
        references.append(ground_truth)

    # Compute the Word Error Rate (WER)
    wer_score = wer_metric.compute(predictions=predictions, references=references)
    return wer_score
```

In [20]:

```python
cv_valid_dev_df['generated_text'] = cv_valid_dev_df['generated_text'].str.lower()

# Evaluate the generated_text output from Task 2 with the groundtruth for cv_valid_dev dataset
wer_score = evaluate_model_generated_text_task2(cv_valid_dev_df)

# Log the overall performance
print(f"Word Error Rate (WER) on the cv_valid_dev set (groundtruth VS generated_text from task 2): {wer_score * 100:.2f}%")
```

```
100%|██████████████████████████████████████████████████| 4076/4076 [00:00<00:00, 17161.3
3it/s]
Word Error Rate (WER) on the cv_valid_dev set (groundtruth VS generated_text from task 2): 11.03%
```

WER evaluates the model's transcription accuracy by comparing predicted transcriptions against ground truth transcriptions. The lower the WER, the better the model performance. If the fine-tuned model performs better on the cv-valid-dev dataset (fine-tuned VS groundtruth), it might indicate that additional fine-tuning on a more comprehensive dataset (cv-valid-test) has improved the model's generalization. If the WER for the fine-tuned model is higher than that of Task 2 (generated text VS groundtruth), it may indicate that the model is overfitting to the cv-valid-test dataset or that the training dataset doesn't match the characteristics of the data in cv-valid-dev.

In this case, the WER on the cv_valid_dev set (groundtruth VS fine-tuned) is 14.39% whereas that on cv_valid_dev set (groundtruth VS generated_text from task 2) is 11.03%. This suggests that the fine-tuned model is overfitting to the cv-valid-test dataset or that the training dataset doesn't match the characteristics of the data in cv-valid-dev. To improve the accuracy, we can implement the following steps:

1. Data augmentation such as speech and volumn pertubation and introducing noise. We can make use of libraries such as torchaudio or librosa and apply these augmentations during training phase.

2. Hyperparameter tuning by adjusting the learning rate (1e-3 to 1e-5) and batch size (8, 16, 32), for instance and pick the one which gave a better performance

3. Training with more epochs if the model has not converged.

4. Regularize the model by including weight decay parameter, for instance.

5. Include more datasets to improve the robustness of the model and to avpid overfitting on the cv-valid-train dataset. One can choose to include open-source datasets from OpenSLR (platform that hosts several open-source datasets related to speech recognition, including datasets for multiple languages), for instance.

In [ ]: