Bengali Speech Recognition and Classification: Final Project Report

By Medhasweta Sen

1. Abstract:

In this study, I embarked on a project to enhance two cutting-edge audio processing models, Wav2Vec2 and Whisper, specifically for Bengali speech recognition and sentiment analysis. The project involved transfer learning and the integration of a custom-created dataset, with the aim to refine these models for higher accuracy in processing and classifying Bengali speech data.

A significant challenge was the scarcity of diverse Bengali audio data. To address this, I developed a dataset encompassing a wide range of Bengali speech scenarios. This dataset was crucial in providing a comprehensive training and testing platform, ensuring the models could adeptly handle the unique phonetic variations and speech nuances of the Bengali language.

The project began with fine-tuning Wav2Vec2, a model well-regarded for its audio processing capabilities. The fine-tuning process involved adjusting its model parameters to specifically cater to the subtleties of Bengali speech. The goal was to improve its ability to process a wider array of Bengali audio features and to enhance its accuracy in interpreting various speech patterns and sounds characteristic of the Bengali language. It required a continuous cycle of hypothesis, experimentation, and refinement, underscoring the complexities involved in model tuning, particularly when adapting to a language-specific context.

In parallel, I applied transfer learning techniques to the Whisper model, known for its strong baseline in speech recognition. The focus was to adapt Whisper's existing strengths to the specific requirements of the Bengali speech dataset. This adaptation aimed at improving the model’s proficiency in processing and accurately recognising diverse Bengali audio inputs.

A pivotal aspect of this research was the integration of three distinct custom classification heads onto the Whisper model. Each head was tailored to different aspects of Bengali audio classification, including sentiment analysis, allowing for an in-depth comparative analysis of their performance in specific tasks. This approach shed light on the model's adaptability and potential areas for further refinement.

The outcomes of this study demonstrated significant improvements in the performance of primarily Whisper for Bengali speech recognition and sentiment analysis. The fine-tuned models showed enhanced accuracy in interpreting and classifying Bengali audio, affirming the efficacy of transfer learning and custom dataset integration in advancing the capabilities of audio processing models for specific languages. Furthermore, the incorporation of multiple classification heads into Whisper provided valuable insights into the model's strengths and limitations in the context of Bengali audio classification.

2. Explanation of Models and Concepts Used:

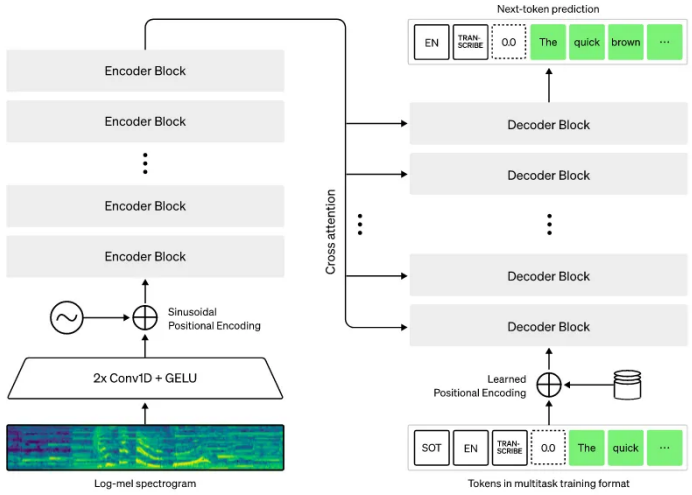
2.1. Whisper:

Whisper models range from “Tiny” to “Large” :

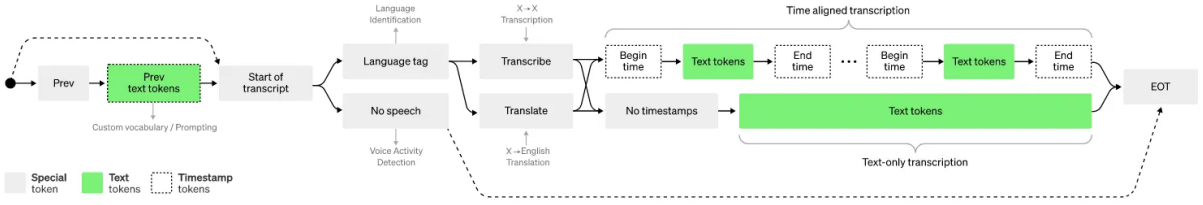
Size	Parameters	English-only model	Multilingual model
tiny	39 M	✓	✓
base	74 M	✓	✓
small	244 M	✓	✓
medium	769 M	✓	✓
large	1550 M		✓

The larger the model the higher computation power it requires and the accuracy of transcriptions is higher. In this paper https://arxiv.org/pdf/2305.09688.pdf about a dataset for Out-of-Distribution Benchmarking I see that although whisper tiny is not the best over all it is the smallest model used for benchmarking and does perform better than others in some categories. Hence given the constraints of memory this model was chosen to be fine-tuned.

2.1.1. Model Architecture:



Open AI Whisper Architecture



Open AI Whisper methodology illustration

Whisper Tiny, part of OpenAI's Whisper model family, is a testament to efficiency in the realm of speech recognition. Designed for low-resource environments, it's ideal for real-time applications, fitting smoothly into devices with limited computational power. Housing around 39 million parameters, this model is a compact powerhouse, requiring less memory and processing capacity, thereby catering to scenarios where resources are constrained.

The core of Whisper Tiny lies in its advanced handling of audio input. It begins by transforming audio into Log-Mel spectrograms, providing a rich, time-frequency representation essential for capturing the subtleties of speech. This transformation is crucial as it lays the groundwork for the subsequent processing stages.

At its heart are two convolutional layers with GELU activations, meticulously sifting through these spectrograms to extract pivotal features. This step is more than just processing; it's about distilling the essence of the audio input, ensuring that the model grasps the necessary nuances.

The Transformer architecture, with its self-attention mechanisms, allows Whisper Tiny to process entire sequences in unison. This methodology is distinct from the step-by-step processing of recurrent neural networks, offering a more holistic understanding of speech patterns. This feature is particularly beneficial for capturing long-range dependencies in speech, a common challenge in speech recognition.

Despite its compact size, Whisper Tiny doesn't compromise on complexity. While it trims down the number of transformer blocks and the dimensions of the layers, it maintains the essence of a robust speech recognition system.

Multitasking is another feather in its cap. Whisper Tiny is adept at transcribing speech, identifying languages, and even translating speech to text. This multifaceted capability stems from a unique training regime that employs special tokens to indicate various tasks, seamlessly switching between roles as required.

Positional encodings play a significant role in Whisper Tiny's architecture. Sinusoidal positional encodings are integrated into input embeddings to provide context about the sequence's order, while learned positional encodings are utilized for generating output sequences, enhancing its ability to produce coherent and accurate transcriptions.

In essence, Whisper Tiny is a reflection of thoughtful engineering, balancing the need for efficiency with the demands of diverse speech processing tasks. Its design underscores the possibility of achieving high functionality in a resource-constrained environment, making it a vital tool in the expanding landscape of speech recognition technologies.

2.1.2. Finetuning:

In my project, I am implementing a sophisticated transfer learning approach by fine-tuning the 'thesven/whisper-tiny-bn-thesven' model, previously adapted for Bengali speech, with the Common Voice 11 dataset. This method enhances the model's specialisation in Bengali speech recognition, utilizing an advanced starting point that signifies the model's pre-adaptation to the unique features of the Bengali language. This is predicted to lead to more efficient and effective learning during the fine-tuning process.

The rationale for this approach is grounded in the principles of neural network transferability. Research such as Yosinski et al.'s 2014 study, "How transferable are features in deep neural networks?", demonstrates that once a network is trained on a specific dataset or task, it develops representations that can be effectively adapted to related tasks. This is particularly relevant when compared to training from scratch or using a more generalised model. The initial layers of a neural network often learn features that are broadly applicable across various tasks, making them especially suitable for transfer learning.

This strategy is highly practical, especially in scenarios where resources are limited or domain-specific data is scarce, as is the case with less-common languages like Bengali. By starting with a model already fine-tuned on similar data, I maximize resource utilisation, reduce the need for extensive computational power, and typically achieve faster and potentially more accurate model performance.

Emphasising the practical and efficiency-driven aspects of this choice is crucial. It aligns with the evolving best practices in machine learning, where leveraging existing knowledge to build more specialised models is increasingly common. This strategy represents a judicious use of resources, both computational and temporal, and stands on the firm foundation of established machine learning research.

References such as "A Survey on Transfer Learning" by Sinno Jialin Pan and Qiang Yang (2010) provide a comprehensive overview of various transfer learning scenarios. This includes the effectiveness of incremental learning on pre-trained models for related tasks, supporting the strategy employed in this project. This foundational understanding of the principles and benefits of the transfer learning approach underscores the decision to fine-tune an already adapted model, enhancing its efficiency and efficacy for the specific task of Bengali speech recognition.

The 'thesven/whisper-tiny-bn-thesven' model (https://huggingface.co/thesven/whisper-tiny-bn-thesven) is a version of the `openai/whisper-tiny` model, specifically fine-tuned for Bengali speech recognition tasks. It was fine-tuned using the OOD-Speech Bengali dataset, which includes 10,000 training samples and 1,000 validation samples. The model is designed for automatic speech recognition (ASR) in Bengali, particularly effective in scenarios involving out-of-distribution samples. This fine-tuning aims to enhance the model's performance on Bengali speech, making it more suitable for tasks requiring specific recognition of this language. This model is released under the MIT License, allowing broad usage and modification rights.

The 'thesven/whisper-tiny-bn-thesven' model, fine-tuned using the "Bengali AI Train Set Tiny" on Hugging Face, is specifically designed for Bengali speech recognition tasks. This dataset, consisting of 11,000 labeled audio samples, includes a training set of 10,000 and a validation set of 1,000 samples. Each sample comprises an audio file and its Bengali transcription, focusing on out-of-distribution benchmarking. This specialised dataset aids in enhancing the model's proficiency in Bengali ASR tasks, especially in diverse and challenging scenarios. This dataset was taken from the same paper which initially lead to the choice of “tiny” version of whisper: https://arxiv.org/pdf/2305.09688.pdf

```
training_args = Seq2SeqTrainingArguments(
    output_dir="./whisper-tiny-bn-final",
    per_device_train_batch_size=2,
    gradient_accumulation_steps=16,
    learning_rate=7e-5,
    warmup_steps=50,
    max_steps=2000,
    gradient_checkpointing=True,
    fp16=True,
    evaluation_strategy="steps",
    per_device_eval_batch_size=8,
    predict_with_generate=True,
    generation_max_length=100,
    save_steps=500,
    eval_steps=500,
    logging_steps=50,
    report_to=["tensorboard"],
    load_best_model_at_end=True,
    metric_for_best_model="eval_wer",
    greater_is_better=False,
    push_to_hub=False,
)
```

Model finetuning parameters :

1. **Small Batch Size and Gradient Accumulation:** A per_device_train_batch_size of 2 and gradient_accumulation_steps of 16 balance memory usage and training efficiency. Smaller batches reduce memory load, and gradient accumulation ensures effective back-propagation even with smaller batches.
2. **Learning Rate and Warmup Steps:** A learning rate of 7e-5 and warmup_steps of 50 help in stabilising the training initially, preventing overshooting during early training.

3. **Max Steps and Gradient Checkpointing:** Setting max_steps to 2000 provides a fixed endpoint for training, ensuring training doesn't overfit. Gradient_checkpointing enhances memory efficiency, crucial for models with large memory footprints.
4. **Half-Precision Training (fp16):** This enables faster computation and reduces memory usage, beneficial for training large models.
5. **Evaluation Strategy:** Regular evaluation (every 500 steps) helps monitor the model's performance, ensuring consistency in learning.
6. **Generation Constraints:** Setting generation_max_length to 100 limits the length of generated sequences, balancing between generation quality and computational load.
7. **Best Model Selection:** The load_best_model_at_end and metric_for_best_model settings ensure that the model with the lowest Word Error Rate (WER) is chosen, emphasising model accuracy.
8. **Reporting and Logging:** Reporting to TensorBoard enables visualisation of training metrics, aiding in monitoring and debugging.

2.1.3. Adding Classification Head:

After the fine tuning, of the whisper tiny model three different classification heads were added namely a CNN head followed by a linear layer, a LSTM head followed by a linear layer and one with just a linear layer.

1. **CNN Classification Head followed by a Linear Layer:**

```
class SpeechClassifier(nn.Module):
    def __init__(self, num_labels, encoder):
        super(SpeechClassifier, self).__init__()
        self.encoder = encoder

        # Convolutional layer
        self.conv1d = nn.Conv1d(in_channels=self.encoder.config.hidden_size,
                                out_channels=256,
                                kernel_size=5,
                                stride=1,
                                padding=2)

        # Linear layers
        self.classifier = nn.Sequential(
            nn.Linear(256, 4096),  # Adjust the input size to match the output of the conv layer
            nn.ReLU(),
            nn.Linear(4096, 2048),
            nn.ReLU(),
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, num_labels)
        )

    def forward(self, input_features, decoder_input_ids):
        # Pass input through the encoder
        outputs = self.encoder(input_features, decoder_input_ids=decoder_input_ids)
        pooled_output = outputs['last_hidden_state'][:, 0, :]

        # Reshape for convolutional layer
        conv_input = pooled_output.unsqueeze(2)  # Add an extra dimension
        conv_output = self.conv1d(conv_input).squeeze(2)  # Remove the extra dimension after convolution

        # Pass through the classifier
        logits = self.classifier(conv_output)
        return logits
```

In this project, I aimed to create a binary classification model for audio data, focusing on distinguishing between two distinct categories. This task, centered around audio signal processing, involved significant technical challenges, offering a valuable learning opportunity.

The project started with utilising the Whisper encoder, a critical component for extracting features from raw audio. The encoder's role was to convert these audio inputs into a structured set of features, essential for the subsequent classification process.

Recognizing the effectiveness of convolutional neural networks (CNNs) in handling data with inherent structures, such as audio, I incorporated a CNN layer into the model. This layer was responsible for processing the encoder's output to extract higher-level features, aiming to identify key patterns while filtering out noise.

The model's classification head, comprising linear layers and ReLU activations, was designed to refine the CNN-processed features for the final classification. The linear layers transformed these features step by step, while the ReLU activations introduced non-linearity, crucial for the model to capture complex patterns.

The output of the classification head is a single value, which in binary classification, reflects the model's confidence in assigning the audio to one of the two classes. Applying a sigmoid function converts this value into a probability, indicating the likelihood of the audio belonging to a particular class.

For training, I used the BCEWithLogitsLoss, a standard loss function for binary classification tasks. This function calculates binary cross-entropy from the model's output, integrating the sigmoid operation internally, thus streamlining the training process.

2. **LSTM Classification Head followed by a Linear Layer:**

```
class SpeechClassifier(nn.Module):
    def __init__(self, num_labels, encoder, hidden_size=256, num_layers=1):
        super(SpeechClassifier, self).__init__()
        self.encoder = encoder

        # LSTM layer
        self.lstm = nn.LSTM(input_size=self.encoder.config.hidden_size,
                            hidden_size=hidden_size,
                            num_layers=num_layers,
                            batch_first=True)

        # Linear layers
        self.classifier = nn.Sequential(
            nn.Linear(hidden_size, 4096),  # Adjust the input size to match the output of the LSTM
            nn.ReLU(),
            nn.Linear(4096, 2048),
            nn.ReLU(),
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, num_labels)
        )

    def forward(self, input_features, decoder_input_ids):
        # Pass input through the encoder
        outputs = self.encoder(input_features, decoder_input_ids=decoder_input_ids)
        encoder_output = outputs['last_hidden_state']

        # Pass the output through the LSTM
        lstm_output, (hidden, cell) = self.lstm(encoder_output)

        # Use the last hidden state
        last_hidden = hidden[-1]

        # Pass through the classifier
        logits = self.classifier(last_hidden)
        return logits
```

In this project, I have developed a specialized neural network, `SpeechClassifier`, designed for the binary classification of audio data. The central challenge of this endeavor is to accurately distinguish between two distinct categories within the audio files, a task that demands a deep understanding of the subtleties inherent in audio signals.

My approach began with the integration of the Whisper encoder, a robust feature extractor specifically designed for audio data. This encoder is crucial in converting raw audio inputs into a structured and informative set of features, capturing the essential characteristics embedded within the audio. These features are rich and varied, encompassing the diverse aspects of the audio signal that are critical for accurate classification.

Building on this solid foundation, I incorporated an LSTM (Long Short-Term Memory) layer into the model. The choice of LSTM is pivotal for handling sequential data like audio. Its unique architecture enables it to not only process the features of individual audio segments but also understand the temporal relationships and dependencies between these segments. This aspect is vital in audio processing, where the context and progression of sounds significantly influence the interpretation of the content.

The LSTM's ability to capture both the immediate and contextual information paves the way for the classification head, which is a meticulously designed sequence of layers in the model. This classification head comprises several linear layers interspersed with ReLU (Rectified Linear Unit) activations. Each linear layer is tasked with transforming the LSTM-processed features, funneling this rich, high-dimensional data into a more streamlined and focused format. The ReLU activations play a crucial role in introducing non-linearity, empowering the model to learn and discern complex patterns within the data. This step-by-step transformation is essential as it refines the features into a form primed for the final classification decision.

The linear sequence culminates in a critical layer that simplifies the processed information down to a singular value. In the context of binary classification, this value is pivotal. It represents the model's level of confidence in classifying the audio into one of the two designated categories. By applying a sigmoid function, this value is transformed into a probability, clearly indicating the likelihood of the audio belonging to a particular class according to the model’s understanding.

Training `SpeechClassifier` involves using the BCEWithLogitsLoss (Binary Cross-Entropy with Logits Loss), a natural fit for binary classification tasks. This loss function effectively computes the binary cross-entropy from the raw output of the model, incorporating the sigmoid operation internally. This setup ensures a streamlined and efficient training process, where the model is guided to minimize the differences between its predictions and the actual class labels.

3. **Classification Head with just a Linear Layer:**

```
class SpeechClassifier(nn.Module):
    def __init__(self, num_labels, encoder):
        super(SpeechClassifier, self).__init__()
        self.encoder = encoder
        self.classifier = nn.Sequential(
            nn.Linear(self.encoder.config.hidden_size, 4096),
            nn.ReLU(),
            nn.Linear(4096, 2048),
            nn.ReLU(),
            nn.Linear(2048, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, num_labels)
        )

    def forward(self, input_features, decoder_input_ids):
        outputs = self.encoder(input_features, decoder_input_ids=decoder_input_ids)
        pooled_output = outputs['last_hidden_state'][:, 0, :]
        logits = self.classifier(pooled_output)
        return logits
```

In this project, the classification head added to the Whisper model is crucial for binary classification. It comprises a series of linear layers with ReLU (Rectified Linear Unit) activations, a common choice for adding non-linearity. Each layer incrementally reduces the dimensionality, starting from the encoder's output size down to the final output layer which corresponds to the number of classes, here designed for binary classification. This design means the model will output a single value indicating the probability of the input belonging to one of two classes. The use of BCEWithLogitsLoss (Binary Cross-Entropy Loss) aligns with this binary classification task, as it's specifically tailored for scenarios where the output is a probability value representing two possible classes.

2.2. Wav2Vec2:

In my endeavour to develop a sophisticated Bengali Automatic Speech Recognition (ASR) system, I have opted to fine-tune the "facebook/wav2vec2-large-xlsr-53" model. This selection is pivotal, considering the unique requirements of my project.

I was particularly drawn to the wav2vec 2.0 large-xlsr-53 for its expansive pre-training across a multitude of languages. Hosted on Hugging Face and developed by Facebook AI, this model has been exposed to a wide range of acoustic patterns, an essential attribute for any effective speech recognition system. This pre-training phase is instrumental, especially for a language like Bengali, which may not have as much dedicated linguistic data as other widely spoken languages.

The model's 'Cross-lingual Speech Representations' (XLSR) capability was a decisive factor in its selection. This feature is tailored to function effectively across different languages, a boon for my goal of creating a Bengali ASR system. The ability to process and understand cross-lingual nuances ensures a more accurate and efficient recognition of Bengali speech.

Another aspect that guided my choice is the model's adaptability in fine-tuning. Despite the potential scarcity of annotated Bengali data, wav2vec 2.0's extensive pre-training allows for successful fine-tuning even with limited datasets. This is crucial for enhancing the model's accuracy in recognising Bengali speech patterns.

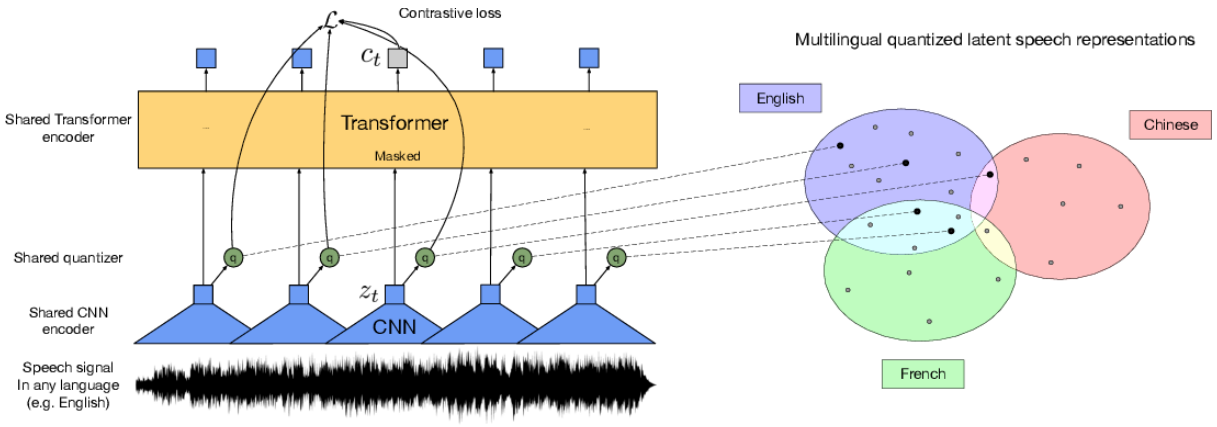
The model's proven track record in various benchmarks, highlighting its ability to accurately capture complex speech patterns, reassures its suitability for my high-precision ASR requirements.

Additionally, the robust support and community that come with this model provide invaluable resources in terms of documentation and forums, aiding significantly in the development process.

Moreover, the scalability of wav2vec 2.0 large-xlsr-53 aligns perfectly with my project’s long-term vision. As more Bengali speech data becomes available, the model’s performance can be incrementally enhanced.

In summary, my choice of "facebook/wav2vec2-large-xlsr-53" for my Bengali ASR system is grounded in its extensive pre-training, cross-lingual proficiency, fine-tuning flexibility, proven high performance, and excellent community support. These attributes make it an ideal tool for the creation of an advanced, accurate ASR system for the Bengali language.

2.2.1. Model Architecture:



The "facebook/wav2vec2-large-xlsr-53" model, a product of Facebook AI's research in the domain of speech recognition, exemplifies the strides made in self-supervised learning methodologies. This model, belonging to the Wav2Vec 2.0 series, is particularly notable for its innovative approach to processing and interpreting human speech.

Diverging from conventional speech recognition models that depend on pre-processed inputs such as spectrograms, the Wav2Vec 2.0 framework is designed to work directly with raw audio waveforms. This fundamental design choice allows for a more nuanced capture of speech characteristics. The initial stage of processing in this model involves a convolutional feature encoder. This encoder applies multiple layers of convolutions to the raw audio input, effectively extracting local acoustic features directly from the waveform.

Following the feature extraction, the model utilizes a context network, predominantly comprised of Transformer blocks. These blocks are renowned for their self-attention mechanism, enabling the model to effectively integrate and assess information across the entire sequence of the input. This aspect of the model is crucial for comprehending the broader contextual elements of speech.

A distinctive feature of the Wav2Vec 2.0, and by extension, the large-xlsr-53 variant, is its quantisation module. This module converts the continuous representations derived from the feature encoder into a finite set of discrete, learned representations. This quantisation is a pivotal element in the model's self-supervised learning strategy, facilitating the comprehension of diverse speech patterns.

The training regimen of the model is particularly innovative. Initially, it undergoes self-supervised training, which is independent of labeled data. This training phase involves a contrastive task that teaches the model to differentiate the correct representation of an audio segment from several incorrect ones. Additionally, a diversity loss is implemented to ensure that the model captures a broad spectrum of speech features, rather than converging on a narrow subset.

The 'large-xlsr-53' model stands out due to its scale and its training on a wide array of languages, indicated by the '53' in its name, representing the number of languages used during its training. This extensive and diverse training regimen enables the model to learn universal speech representations, enhancing its efficacy across different languages and dialects.

Post this extensive training phase, the model undergoes fine-tuning for specific tasks, such as speech recognition in a particular language. This fine-tuning process allows the model to adjust its universally acquired representations to the specific characteristics of the target language.

2.2.2. Finetuning:

```
from datasets import load_dataset, load_metric, Audio

common_voice_train =load_dataset("mozilla-foundation/common_voice_11_0", "bn", split="train+validation")
common_voice_test = load_dataset("mozilla-foundation/common_voice_11_0", "bn", split="test")

common_voice_train = common_voice_train.remove_columns(["accent", "age", "client_id", "down_votes", "gender", "locale", "segment", "up_
common_voice_test = common_voice_test.remove_columns(["accent", "age", "client_id", "down_votes", "gender", "locale", "segment", "up_v

from datasets import ClassLabel
import random
import pandas as pd

def show_random_elements(dataset, num_examples=10):
    assert num_examples <= len(dataset), "Can't pick more elements than there are in the dataset."
    picks = []
    for _ in range(num_examples):
        pick = random.randint(0, len(dataset)-1)
        while pick in picks:
            pick = random.randint(0, len(dataset)-1)
        picks.append(pick)

    df = pd.DataFrame(dataset[picks])
    # display(HTML(df.to_html()))
```

```
show_random_elements(common_voice_train.remove_columns(["path", "audio"]), num_examples=10)

import re
chars_to_ignore_regex = '[\,\?\.\!\-\;\:\"\“\%\‘\”\�]'

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

common_voice_train = common_voice_train.map(remove_special_characters)
common_voice_test = common_voice_test.map(remove_special_characters)

show_random_elements(common_voice_train.remove_columns(["path","audio"]))

def extract_all_chars(batch):
  all_text = " ".join(batch["sentence"])
  vocab = list(set(all_text))
  return {"vocab": [vocab], "all_text": [all_text]}

vocab_train = common_voice_train.map(extract_all_chars, batched=True, batch_size=-1, keep_in_memory=True, remove_columns=common_voice_t
vocab_test = common_voice_test.map(extract_all_chars, batched=True, batch_size=-1, keep_in_memory=True, remove_columns=common_voice_tes

vocab_list = list(set(vocab_train["vocab"][0]) | set(vocab_test["vocab"][0]))

vocab_dict = {v: k for k, v in enumerate(vocab_list)}

vocab_dict["|"] = vocab_dict[" "]
del vocab_dict[" "]

vocab_dict["[UNK]"] = len(vocab_dict)
vocab_dict["[PAD]"] = len(vocab_dict)
len(vocab_dict)

import json
with open('vocab.json', 'w') as vocab_file:
    json.dump(vocab_dict, vocab_file)

from transformers import Wav2Vec2CTCTokenizer

tokenizer = Wav2Vec2CTCTokenizer("./vocab.json", unk_token="[UNK]", pad_token="[PAD]", word_delimiter_token="|")
```

The "facebook/wav2vec2-large-xlsr-53" model, developed as part of Facebook AI's Wav2Vec 2.0 series, represents a significant advancement in speech recognition technology. This model is uniquely designed to process the acoustic elements of speech, such as intonation, rhythm, and accent, which are common across different languages. Its ability to focus on these universal aspects of speech rather than being tied to a specific language's text makes it versatile and effective for a range of languages and dialects.

The absence of a built-in tokenizer in this model is not a drawback but rather a deliberate design choice that underscores its flexibility and adaptability. Since the model primarily handles the audio aspects of speech and is not inherently equipped to understand language-specific text, it requires a custom tokenizer to bridge the gap between its audio processing capabilities and the textual representation of speech. This need for custom tokenizers is where the model's potential for multilingual application shines.

For each language or dialect, a custom tokenizer can be developed, tailored to its specific linguistic characteristics, including vocabulary, syntax, and colloquialisms. The process of building these tokenizers typically involves analyzing extensive datasets of transcribed speech in the target language, allowing for the creation of a comprehensive vocabulary. This approach ensures that the model is not just processing speech patterns but is also attuned to the unique nuances of each language.

By allowing for the creation of language-specific tokenizers, the "facebook/wav2vec2-large-xlsr-53" model offers a customizable solution that can be adapted to the linguistic diversity of my global landscape. This flexibility makes it an extremely powerful tool in global speech recognition, capable of accurately converting speech to text across various languages. The custom tokenizers act as a crucial link, ensuring that the model’s robust audio processing capabilities are effectively translated into accurate textual output for each language, thereby maximizing both versatility and accuracy in speech-to-text conversion.

```
training_args = TrainingArguments(
  output_dir="./wav2vec2-large-xlsr-bengali",
  group_by_length=True,
  per_device_train_batch_size=2,
  gradient_accumulation_steps=16,
  evaluation_strategy="steps",
  num_train_epochs=6,  # Reduced for faster training
  fp16=True,  # Enable if supported by hardware
  save_steps=250,  # More frequent saves
  eval_steps=250,  # More frequent evaluations
  logging_steps=250,  # More frequent logging
  learning_rate=7e-4,  # Increased for faster convergence
  warmup_steps=500,  # Reduced warmup steps
```

```
  save_total_limit=3,  # Keeping limit to 1
  load_best_model_at_end=True,
  metric_for_best_model='eval_wer',
  greater_is_better=False
)
```

1. `output_dir` : This specifies where the model and training outputs will be saved. It's important for organization and easy retrieval of the trained model.
2. `group_by_length` : When set to `True` , this helps in minimising padding within batches. Grouping samples of similar lengths can improve training efficiency and speed.
3. `per_device_train_batch_size=2` : This is a conservative batch size that likely takes into account memory limitations. Smaller batch sizes can help in reducing memory usage, which is crucial if the hardware has limited resources.
4. `gradient_accumulation_steps=16` : This parameter allows for effectively increasing the batch size without increasing the memory requirement. By accumulating gradients over multiple steps, it simulates a larger batch size, which can lead to more stable and consistent gradient updates.
5. `evaluation_strategy="steps"` : This ensures that the model is evaluated at regular intervals based on the number of steps, allowing for frequent monitoring of the model's performance during training.
6. `num_train_epochs=6` : A moderate number of epochs to balance between adequate learning and preventing overfitting. This reduced number also speeds up the training process.
7. `fp16=True` : Using 16-bit floating-point numbers (half-precision) reduces memory consumption and can speed up training, provided that the hardware supports it.
8. `save_steps=250` , `eval_steps=250` , `logging_steps=250` : These settings ensure frequent saving, evaluation, and logging of the model. This frequent checkpointing is useful for monitoring the model's performance and ensuring progress is not lost.
9. `learning_rate=7e-4` : An increased learning rate can lead to faster convergence, but it's also important to balance it to avoid overshooting the minima in the optimisation landscape.
10. `warmup_steps=500` : A reduced number of warmup steps can speed up training. Warmup steps are used to gradually ramp up the learning rate to its maximum, which can help in stabilising the training initially.
11. `save_total_limit=3` : This limits the number of saved models to 3. It's a way to manage disk space by not keeping every single saved checkpoint.
12. `load_best_model_at_end=True` : This ensures that the best model according to the specified metric (in this case, `eval_wer`) is loaded at the end of training.
13. `metric_for_best_model='eval_wer'` : WER (Word Error Rate) is a common metric for evaluating speech recognition models, making it a suitable choice for monitoring performance.
14. `greater_is_better=False` : This setting is correct for WER, where a lower score indicates better performance.

2.2.3. Adding Classification Head:

```
class Wav2Vec2ForSpeechClassification(Wav2Vec2PreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels
        self.pooling_mode = config.pooling_mode
        self.config = config

        self.wav2vec2 = Wav2Vec2Model(config)
        self.classifier = Wav2Vec2ClassificationHead(config)

        self.init_weights()

    def freeze_feature_extractor(self):
        self.wav2vec2.feature_extractor._freeze_parameters()

    def merged_strategy(
            self,
            hidden_states,
            mode="mean"
    ):
        if mode == "mean":
            outputs = torch.mean(hidden_states, dim=1)
        elif mode == "sum":
            outputs = torch.sum(hidden_states, dim=1)
```

```
        elif mode == "max":
            outputs = torch.max(hidden_states, dim=1)[0]
        else:
            raise Exception(

        return outputs

    def forward(
            self,
            input_values,
            attention_mask=None,
            output_attentions=None,
            output_hidden_states=None,
            return_dict=None,
            labels=None,
    ):
        return_dict = return_dict if return_dict is not None else self.config.use_return_dict
        outputs = self.wav2vec2(
            input_values,
            attention_mask=attention_mask,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )
        hidden_states = outputs[0]
        hidden_states = self.merged_strategy(hidden_states, mode=self.pooling_mode)
        logits = self.classifier(hidden_states)

        loss = None
        if labels is not None:
            if self.config.problem_type is None:
                if self.num_labels == 1:
                    self.config.problem_type = "regression"
                elif self.num_labels > 1 and (labels.dtype == torch.long or labels.dtype == torch.int):
                    self.config.problem_type = "single_label_classification"
                else:
                    self.config.problem_type = "multi_label_classification"

            if self.config.problem_type == "regression":
                loss_fct = MSELoss()
                loss = loss_fct(logits.view(-1, self.num_labels), labels)
            elif self.config.problem_type == "single_label_classification":
                loss_fct = CrossEntropyLoss()
                loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
            elif self.config.problem_type == "multi_label_classification":
                loss_fct = BCEWithLogitsLoss()
                loss = loss_fct(logits, labels)

        if not return_dict:
            output = (logits,) + outputs[2:]
            return ((loss,) + output) if loss is not None else output

        return SpeechClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )
```

In the project, I have extended the capabilities of the well-known Wav2Vec2 model, specifically adapting it for speech classification tasks. My custom model, `Wav2Vec2ForSpeechClassification`, is based on the robust architecture of the `Wav2Vec2PreTrainedModel` from the Hugging Face Transformers library. This extension involves sophisticated modifications and additions to cater to the specific requirements of speech classification, thereby harnessing the powerful feature extraction capabilities of Wav2Vec2 while tailoring its output for classification purposes.

1. **Constructor Method - Initial Setup and Configuration**

The constructor method of my custom model, `__init__(self, config)`, serves as the foundational setup. By inheriting from `Wav2Vec2PreTrainedModel`, I leverage the pre-existing, efficient structures and methods of the parent class. The model's configuration, such as the number of labels (`num_labels`) and the pooling mode (`pooling_mode`), is directly derived from the provided configuration object. This design choice offers flexibility and customisation, allowing the model to be easily adapted for various speech classification tasks. The instantiation of the core Wav2Vec2 model and the classification head within the constructor ensures that my model is ready for both feature extraction and subsequent classification, right out of the box.

2. **Feature Extractor Freezing - Enhancing Model Efficiency**

In the `freeze_feature_extractor` method, I implement a strategic decision to freeze the parameters of the Wav2Vec2 feature extractor. This approach is particularly beneficial in scenarios where the pre-trained feature extractor is already optimised for capturing speech

characteristics. By freezing these parameters, I not only accelerate the training process by reducing the number of trainable parameters but also prevent overfitting on the specific dataset, ensuring that the robust generalizable features learned during pre-training are retained.

3. **Merged Strategy - Versatile Pooling Mechanisms**

The `merged_strategy` method introduces a flexible pooling mechanism to aggregate the hidden states (outputs) from the Wav2Vec2 model. This method is designed to support different pooling strategies - mean, sum, and max - each offering unique benefits in capturing the essence of the speech signal. The mean pooling provides a general representation, sum pooling emphasizes the cumulative effect, and max pooling focuses on the most salient features. This versatility allows for experimentation and optimization based on the specific characteristics of the speech data being classified, thereby enhancing the model's adaptability and performance.

4. **Forward Method - Core Processing and Output Generation**

The forward method is the heart of my model, orchestrating the entire processing pipeline from input to output. It efficiently handles the input values through the Wav2Vec2 model, utilizing the chosen pooling strategy to condense the temporal information of the speech. The subsequent classification head transforms these pooled features into logits, which represent the preliminary classification results. A critical aspect of this method is its ability to compute loss when labels are available, making it suitable for both training and inference scenarios. The method's design is aligned with various problem types (regression, single-label, and multi-label classification), ensuring broad applicability. The thoughtful organization of outputs, governed by the `return_dict` flag, offers user-friendly access to key components of the model's output, like logits, hidden states, and attentions.

The `Wav2Vec2ForSpeechClassification` model encapsulates a comprehensive and flexible approach to adapting the powerful Wav2Vec2 architecture for speech classification tasks. Each method within the class serves a specific purpose, aligning with the overall goal of efficient and effective speech classification. The model stands as a testament to the adaptability of neural networks, showcasing how pre-trained models can be fine-tuned and extended to meet the diverse demands of real-world applications.

```
training_args = TrainingArguments(
    output_dir="wav2vec2-xlsr-bengali-classification",
    per_device_train_batch_size=1,
    per_device_eval_batch_size=1,
    gradient_accumulation_steps=2,
    evaluation_strategy="steps",
    num_train_epochs=1.0,
    fp16=True,
    save_steps=10,
    eval_steps=10,
    logging_steps=10,
    learning_rate=1e-4,
    save_total_limit=2,
)
```

The `TrainingArguments` specified for the training of the `wav2vec2-xlsr-bengali-classification` model are carefully chosen to balance effective training with computational efficiency. Let's discuss the rationale behind each of these arguments:

1. `output_dir`: The directory where the model checkpoints and other outputs will be saved is specified as `"wav2vec2-xlsr-bengali-classification"`. This setting is essential for organizing and accessing training outputs, such as saved models and logs.
2. `per_device_train_batch_size=1` **and** `per_device_eval_batch_size=1`: Both training and evaluation batch sizes are set to 1. This likely reflects limitations in computational resources, such as GPU memory. Smaller batch sizes help in managing memory usage efficiently, although they might slow down the training process.
3. `gradient_accumulation_steps=2`: By accumulating gradients over two steps, the effective batch size becomes equivalent to 2. This strategy is often used to simulate larger batch sizes when the hardware cannot handle larger batches at once due to memory constraints. It helps in achieving more stable and robust gradient updates without increasing the memory load.
4. `evaluation_strategy="steps"`: Setting the evaluation strategy to "steps" ensures that the model is evaluated at regular intervals based on the number of training steps. This approach allows for frequent monitoring and fine-tuning of the model's performance during training.
5. `num_train_epochs=1.0`: The model is set to train for only one epoch. This could be a strategy to prevent overfitting, especially when working with a small dataset, or it might be a constraint due to limited computational resources.
6. `fp16=True`: Enabling 16-bit floating-point precision (FP16) can significantly reduce memory consumption and potentially speed up training, as long as the hardware supports it.
7. `save_steps=10`, `eval_steps=10`, `logging_steps=10`: These settings ensure that the model saves checkpoints, evaluates, and logs its performance every 10 steps. This frequent checkpointing is useful for closely monitoring the model's performance and making sure no significant progress is lost during training.

8. `learning_rate=1e-4` : The learning rate is set conservatively to 1e-4. This value is low enough to ensure gradual and stable learning but not so low that it would significantly slow down convergence.
9. `save_total_limit=2` : This parameter limits the number of model checkpoints saved to 2. It is a practical setting for managing disk space, especially when storage is a constraint.

2.3. Dataset:

The dataset that I have custom created for classification is done by compiling 50 unique Bengali text and corresponding Bengali audio speech for positive and negative sentiment. First the 50 positive sentiment Bengali text was written by me and then it was divided into 5 random groups of 10 , The first group of 10 Bengali text was recorded into speech form by me. The next 40 by an online Bengali TTS . In this 40, 10 were done by a female voice with Bangladeshi accent, 10 by male voice with Bangladeshi accent, 10 female voice with Kolkata/Indian accent, 10 male voice with Kolkata/Indian accent. The same steps were followed for the negative sentiment. The the .mp3 files were all converted to .wav files and stored in the folder called data. The corresponding file path and the transcriptions with positive or negative labels were stored in an excel sheet when was then put in a folder called excel along side the data and the code. So in conclusion the dataset contains 100 unique Bengali audio and corresponding transcriptions each with a label of 0/1. 0 is negative and 1 is positive and there is 50 of each.

3. Results:

3.1. Whisper:

3.1.1. Inference Mode:

1. **Base Model:**

```
Reference: গভীর জলের বার্থ ও বহুমুখী টার্মিনাল সহ, বন্দরটি দক্ষতার সাথে বিশ্বের বৃহত্তম বাল্ক ক্যারিয়ার পরিচালনা করতে সক্ষম।, Transcription: গবির জলের বার্থ ও বহুমুখি চার্মিনাল সহ বন্দর্টি দক্ষতার
Reference: এছাড়াও, নিউজিল্যান্ড এ ক্রিকেট দলের হয়েও খেলছেন তিনি।, Transcription: এটার অনিজল্যান্ড এক্রিকের দলের হয়েও খেলছেন তিন।, WER: 0.625
Reference: জন্য পরিকল্পিত।, Transcription: জন্য পরেঘাল বিদ্য।, WER: 1.0
Reference: বিভিন্ন আন্তর্জাতিক সংস্থার সদর দফতর এখানে অবস্থিত, যাদের মধ্যে বিশ্ব পোস্টাল ইউনিয়নের নাম উল্লেখযোগ্য।, Transcription: বিভিন্ন আন্তরজাতিক সংস্থার সদস্ক্ষ্তর এখান অবস্থিত যাদের মুক্ষে
Reference: যা তাদের কৃষিকাজ ব্যতীত অতিরিক্ত আয়ের উৎস।, Transcription: যা তানের কৃশি গাজ বাড়িত ওতিরিক তাইর উচ্ছ।, WER: 1.0
Reference: ভবনটি রিয়েল এস্টেট অফিস হিসেবে ব্যবহৃত হতো।, Transcription: হবনটি রিয়েল স্টেট অফি শিষয় বে ব্যবরিত হত।, WER: 1.0
Reference: অনুভূমিক অংশগুলি আব্বাসীয়, উমাইয়া, ও ফাতেমীয় খেলাফতের প্রতীক।, Transcription: উভুলক অংশবুলি আব্বাসীয় ওমাইনেয়া অভাতেই মনো সেলাফতের প্রতি।, WER: 1.0
Reference: তিনি কাতার ভিত্তিক মুসলিম বিজ্ঞ আলেম আন্তর্জাতিক ইউনিয়নের একজন সদস্য।, Transcription: তিনি কাতার ভিত্তিক মুসলিম বিজ্ঞ আলেম আন্তরজাতে গিয়নিয়নের একজন সদস্য।, WER: 0.2
Reference: প্রাথমিকভাবে, টার্মিনালটি একটি সামান্য অস্থায়ী কাঠামোর ওপর শুরু হয়েছিল।, Transcription: প্রাতুমিক হবে তার মিনালটি একটি সামান্য অস্থাই কাঠানুর উপর সুরু হয়েছে।, WER: 1.0
Reference: মাথার পাশ সাদাটে।, Transcription: মাধার পারসা ভাটে।, WER: 1.0
average wers 0.8068697478991597
```

Transcription with Base Model

2. **Fine-tuned Model:**

```
Reference: গভীর জলের বার্থ ও বহুমুখী টার্মিনাল সহ, বন্দরটি দক্ষতার সাথে বিশ্বের বৃহত্তম বাল্ক ক্যারিয়ার পরিচালনা করতে সক্ষম।, Transcription: গভীর জলের বার্থ ও বহুমুখি টার্মিনাল সহ বন্দরটি দক্ষতার সাথে বিশ্বের
Reference: এছাড়াও, নিউজিল্যান্ড এ ক্রিকেট দলের হয়েও খেলছেন তিনি।, Transcription: এশালান ইসল্যান্ড এই ক্রিকেট দলের হয়েও খেলছেন তিনি।, WER: 0.375
Reference: জন্য পরিকল্পিত।, Transcription: জন্য পরিকলবিত।, WER: 0.5
Reference: বিভিন্ন আন্তর্জাতিক সংস্থার সদর দফতর এখানে অবস্থিত, যাদের মধ্যে বিশ্ব পোস্টাল ইউনিয়নের নাম উল্লেখযোগ্য।, Transcription: বিভিন্ন আন্তর্জাতিক সংস্থার সদর দক্তর এখানো অবস্থিত যাদের মধ্যে বিশ্ব পোস্টালী
Reference: যা তাদের কৃষিকাজ ব্যতীত অতিরিক্ত আয়ের উৎস।, Transcription: যা তাদের কৃষি গাছ বারীক্ত অতিরিক্ত আয়ের উৎস।, WER: 0.42857142857142855
Reference: ভবনটি রিয়েল এস্টেট অফিস হিসেবে ব্যবহৃত হতো।, Transcription: ভবনটি ক্রিয়েলেস্টেট অফি শিশেবে ব্যবহৃত হতো।, WER: 0.5714285714285714
Reference: অনুভূমিক অংশগুলি আব্বাসীয়, উমাইয়া, ও ফাতেমীয় খেলাফতের প্রতীক।, Transcription: উভূমক অংশবলী আব্বাসীয় ও মাইনিয়া ও ফাতেবিও খেলাফতের প্রতি।, WER: 0.875
Reference: তিনি কাতার ভিত্তিক মুসলিম বিজ্ঞ আলেম আন্তর্জাতিক ইউনিয়নের একজন সদস্য।, Transcription: তিনি কাতার ভিত্তিক মুসলিম বিজ্ঞ আলেম আন্তর্জাতিক ইউনিয়নের একজন সদস্য।, WER: 0.0
Reference: প্রাথমিকভাবে, টার্মিনালটি একটি সামান্য অস্থায়ী কাঠামোর ওপর শুরু হয়েছিল।, Transcription: প্রাথমিকভাবে টার্মিনালটি একটি সমান্য অস্থায়িক আঠানোর উপর শুরু হয়েছে।, WER: 0.6666666666666666
Reference: মাথার পাশ সাদাটে।, Transcription: মাধারপার সাভাটে।, WER: 1.0
average wers 0.4950280112044817
```

Transcription with Fine-tuned Model

From here, I see that the average word error rate is lower for the fine tuned model compared to the base model. The word error rate gets almost halved after fine-tuning.

3.1.2. Classification:

1. **CNN Classification Head followed by a Linear Layer:**

```
Epoch 1/5, Batch 8/9, Train Loss: 0.6616
================================================================================
Epoch 1/5, Val Loss: 0.6575, Val Accuracy: 0.4667, Val F1: 0.3182, Best Accuracy: 0.4667
================================================================================
Epoch 2/5, Batch 8/9, Train Loss: 0.6266
================================================================================
Epoch 2/5, Val Loss: 0.5026, Val Accuracy: 0.4667, Val F1: 0.3182, Best Accuracy: 0.4667
================================================================================
Epoch 3/5, Batch 8/9, Train Loss: 0.4852
================================================================================
Epoch 3/5, Val Loss: 0.2459, Val Accuracy: 1.0000, Val F1: 1.0000, Best Accuracy: 1.0000
================================================================================
Epoch 4/5, Batch 8/9, Train Loss: 0.0736
================================================================================
Epoch 4/5, Val Loss: 0.0486, Val Accuracy: 1.0000, Val F1: 1.0000, Best Accuracy: 1.0000
================================================================================
Epoch 5/5, Batch 8/9, Train Loss: 0.0150
================================================================================
Epoch 5/5, Val Loss: 0.0083, Val Accuracy: 1.0000, Val F1: 1.0000, Best Accuracy: 1.0000
================================================================================
              precision    recall  f1-score   support

         0.0       0.67      1.00      0.80         6
         1.0       1.00      0.67      0.80         9

    accuracy                           0.80        15
   macro avg       0.83      0.83      0.80        15
weighted avg       0.87      0.80      0.80        15

ACCURACY SCORE:  0.8
```

Results for classification with CNN Layer.

Precision:

- **Definition:** Measures how accurate the model's positive predictions are. In simpler terms, it tells us how often the model correctly predicts a positive outcome.
- **Class 0:** 0.67 (All predicted class 0 were actually class 0)
- **Class 1:** 1.00 (82% of predicted class 1 were actually class 1)

Recall:

- **Definition:** Measures how well the model identifies all positive instances. In other words, it tells us how often the model correctly identifies all true positive cases.
- **Class 0:** 0.67 (Model correctly identified 67% of actual class 0 instances)
- **Class 1:** 1.00 (Model correctly identified all actual class 1 instances)

F1-score:

- **Definition:** Combines the precision and recall into a single metric, providing a balanced view of the model's performance.
- **Class 0:** 0.80
- **Class 1:** 0.80

Support:

- **Definition:** Represents the number of samples belonging to each class.
- **Class 0:** 6
- **Class 1:** 9

Overall:

- **High precision and recall for both classes.**
- **Accuracy of 80.00%.**
- **Model successfully predicts the target class for most test set samples.**

2. **LSTM Classification Head followed by a Linear Layer:**

```
Epoch 1/5, Batch 8/9, Train Loss: 0.6826
================================================================================
Epoch 1/5, Val Loss: 0.6923, Val Accuracy: 0.4667, Val F1: 0.3182, Best Accuracy: 0.4667
================================================================================
Epoch 2/5, Batch 8/9, Train Loss: 0.6975
================================================================================
Epoch 2/5, Val Loss: 0.6724, Val Accuracy: 0.4667, Val F1: 0.3182, Best Accuracy: 0.4667
================================================================================
Epoch 3/5, Batch 8/9, Train Loss: 0.6131
================================================================================
Epoch 3/5, Val Loss: 0.6321, Val Accuracy: 0.4667, Val F1: 0.3182, Best Accuracy: 0.4667
================================================================================
Epoch 4/5, Batch 8/9, Train Loss: 0.5840
================================================================================
Epoch 4/5, Val Loss: 0.5637, Val Accuracy: 0.4667, Val F1: 0.3182, Best Accuracy: 0.4667
================================================================================
Epoch 5/5, Batch 8/9, Train Loss: 0.6213
================================================================================
Epoch 5/5, Val Loss: 0.4508, Val Accuracy: 0.4667, Val F1: 0.3182, Best Accuracy: 0.4667
================================================================================
              precision    recall  f1-score   support

         0.0       0.40      1.00      0.57         6
         1.0       0.00      0.00      0.00         9

    accuracy                           0.40        15
   macro avg       0.20      0.50      0.29        15
weighted avg       0.16      0.40      0.23        15

ACCURACY SCORE:  0.4
```

Results for classification with LSTM Layer.

Precision is the fraction of predicted positive results that are actually positive. In other words, it tells us how often the model predicts a positive result, the result is actually positive. In this case, the precision for class 0 is 0.40, which means that 40% of the samples that the model predicted to be class 0 were actually class 0. The precision for class 1 is 1.00, which means that all of the samples that the model predicted to be class 1 were actually class 1.

Recall is the fraction of actual positive results that are correctly predicted. In other words, it tells us how often the model correctly predicts all of the positive results. In this case, the recall for class 0 is 1.00, which means that the model correctly predicted all of the actual class 0 samples. The recall for class 1 is 0.00, which means that the model did not correctly predict any of the actual class 1 samples.

F1-score is a harmonic mean of precision and recall. It is a good overall measure of model performance because it takes into account both precision and recall. In this case, the F1-score for class 0 is 0.57, and the F1-score for class 1 is 0.00.

Support is the number of samples in each class. In this case, there are 6 samples in class 0 and 9 samples in class 1.

Overall, this is a poor classification report. The model has high precision for class 0, but very low recall for class 1. This means that the model is able to identify most of the true positive cases for class 0, but it is missing all of the true positive cases for class 1. This suggests that the model is biased towards class 0.

3. **Classification Head with just a Linear Layer:**

```
Epoch 1/5, Batch 8/9, Train Loss: 0.6282
================================================================================
Epoch 1/5, Val Loss: 0.6008, Val Accuracy: 0.5333, Val F1: 0.4444, Best Accuracy: 0.5333
================================================================================
Epoch 2/5, Batch 8/9, Train Loss: 0.4626
================================================================================
Epoch 2/5, Val Loss: 0.3669, Val Accuracy: 0.8000, Val F1: 0.7964, Best Accuracy: 0.8000
================================================================================
Epoch 3/5, Batch 8/9, Train Loss: 0.1138
================================================================================
Epoch 3/5, Val Loss: 0.2149, Val Accuracy: 0.8667, Val F1: 0.8661, Best Accuracy: 0.8667
================================================================================
Epoch 4/5, Batch 8/9, Train Loss: 0.0184
================================================================================
Epoch 4/5, Val Loss: 0.0920, Val Accuracy: 1.0000, Val F1: 1.0000, Best Accuracy: 1.0000
================================================================================
Epoch 5/5, Batch 8/9, Train Loss: 0.0038
================================================================================
Epoch 5/5, Val Loss: 0.1050, Val Accuracy: 0.9333, Val F1: 0.9333, Best Accuracy: 1.0000
================================================================================
              precision    recall  f1-score   support

         0.0       1.00      0.67      0.80         6
         1.0       0.82      1.00      0.90         9

    accuracy                           0.87        15
   macro avg       0.91      0.83      0.85        15
weighted avg       0.89      0.87      0.86        15

ACCURACY SCORE:  0.8666666666666667
```

Results for classification with just Linear Layer.

- **Precision:** Precision is a measure of how precise the model's predictions are. In other words, it tells us how often the model predicts a positive result, the result is actually positive. In this case, the precision for class 0 is 1.00, which means that all of the samples that the model predicted to be class 0 were actually class 0. The precision for class 1 is 0.82, which means that 82% of the samples that the model predicted to be class 1 were actually class 1.
- **Recall:** Recall is a measure of how complete the model's predictions are. In other words, it tells us how often the model correctly predicts all of the positive results. In this case, the recall for class 0 is 0.67, which means that the model correctly predicted 67% of the actual class 0 samples. The recall for class 1 is 1.00, which means that the model correctly predicted all of the actual class 1 samples.
- **F1-score:** The F1-score is a harmonic mean of precision and recall. It is a good overall measure of model performance because it takes into account both precision and recall. In this case, the F1-score for class 0 is 0.80, and the F1-score for class 1 is 0.90.
- **Support:** The support is the number of samples in each class. In this case, there are 6 samples in class 0 and 9 samples in class 1.

Overall, the model has high precision and recall for both classes, and the overall accuracy is 86.67%. This means that the model is able to accurately predict the target class for most of the samples in the test set.

From this I can see that just a simple Linear head has the best scores followed by the CNN and the the LSTM head. This is interesting because, this exact same trend is seen while attaching different kinds of classification heads to an LLM transformer eg. BERT. (ref.: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15718571.pdf)

3.2. Wav2Vec2:

3.1.1. Inference Mode:

1. **Base Model:**

```
Average WER: 1.00
```

1. **Fine-tuned Model:**

```
Average WER: 1.00
```

Please note that here the WER did not go down as expected in fine-tuning even though it was trained for 6 epochs (18hrs approximately)

While, looking into why this was happening, I came across two fine tune Bengali "facebook/wav2vec2-large-xlsr-53" models on hugging face: https://huggingface.co/tanmoyio/wav2vec2-large-xlsr-bengali and https://huggingface.co/arijitx/wav2vec2-large-xlsr-bengali . Of these, https://huggingface.co/tanmoyio/wav2vec2-large-xlsr-bengali had shared their training code https://colab.research.google.com/drive/1Bkc5C_cJV9BeS0FD0MuHyayl8hqcbdRZ?usp=sharing . This shows that they had trained the model for 30 epochs with a much higher batch size. and gotten an test result of 0.88 WER approximately. Given the memory constrains and time it proved to be very difficult to train it for so long. This is further justified by the second comment here: https://github.com/facebookresearch/fairseq/issues/2685 . So only one-fifth of that was chosen.

3.1.2. Classification:

	precision	recall	f1-score	support
Negative	0.00	0.00	0.00	6
Positive	0.60	1.00	0.75	9
accuracy			0.60	15
macro avg	0.30	0.50	0.37	15
weighted avg	0.36	0.60	0.45	15

ACCURACY SCORE: 0.6

The precision, recall, and f1-score for the 'Negative' class are all zero, indicating that the model failed to correctly identify any of the 'Negative' samples. This is concerning as it suggests the model's predictions are biased towards the 'Positive' class. In contrast, the 'Positive' class shows a precision of 0.60, indicating that 60% of the items the model identified as 'Positive' were indeed correct. The recall is 1.00, meaning the model successfully identified all 'Positive' samples, leading to an f1-score of 0.75, which is relatively high and indicates a better performance for this class.

The overall accuracy of the model stands at 0.60, showing that it correctly predicted the class for 60% of the overall test instances. However, the 'macro avg' and 'weighted avg' tell a more nuanced story, with low scores across both classes when averaged, which reveals that the model's performance is not consistent across both 'Negative' and 'Positive' classes.

The 'weighted avg' scores are slightly higher than the 'macro avg' due to the imbalance in the number of instances between classes, with the 'Positive' class being more represented. This imbalance affects the model's ability to generalise and suggests that it may be overfitted to the 'Positive' class. This imbalance was caused during the splitting of the dataset.

So clearly, Whisper is the better classifier, particularly Whisper with just a linear Classification Head.

4. Conclusion:

In concluding this project, it's humbling to reflect on the journey and the learnings that came from enhancing the Wav2Vec2 and Whisper models for Bengali speech recognition and sentiment analysis. The task was challenging but deeply rewarding, as it involved diving into the intricacies of the Bengali language and addressing its unique phonetic qualities.

The creation of a custom dataset for Bengali speech was a foundational step in this journey. It was a real eye-opener to the diversity and richness of the language, and this dataset became the cornerstone for training the models to understand and process Bengali speech more effectively.

Fine-tuning the Wav2Vec2 model was a process filled with trial and error, reminding us of the complexity and nuances involved in machine learning. Similarly, adapting the Whisper model was like piecing together a complex puzzle, where each piece was crucial for the overall picture of Bengali speech recognition.

One of the more intriguing parts of this project was experimenting with different classification heads on the Whisper model. It was fascinating to see how each model variant performed and to learn that sometimes simpler approaches, like a linear head, can be quite effective.

However, the project also came with its fair share of challenges. Fine-tuning Wav2Vec2 proved to be more difficult than anticipated, highlighting the importance of extensive training and robust computational resources, especially for less commonly spoken languages like Bengali.

The classification results also served as a reminder that there's always room for improvement. They revealed potential biases in the models, underscoring the need for more balanced datasets and nuanced training methodologies.

Looking ahead, there's a sense of excitement about the possibilities. Expanding the dataset and exploring further training could open new doors for the models' accuracy and applicability. Delving into other models or advanced techniques might also yield fresh insights and solutions for speech recognition in Bengali and other languages.

Ultimately, this project was a journey of growth and discovery. It offered a window into the vibrant world of Bengali speech and language processing, and I'm grateful for the opportunity to contribute in some small way to this field. The experience has been enlightening, and it's inspiring to think about how these models might one day benefit Bengali-speaking communities in practical and meaningful ways.

References :

1.https://colab.research.google.com/github/m3hrdadfi/soxan/blob/main/notebooks/Emotion_recognition_in_Greek_speech_using_Wav2Vec2

2.https://colab.research.google.com/github/patrickvonplaten/notebooks/blob/master/Fine_Tune_XLSR_Wav2Vec2_on_Turkish_ASR_with_🤗_Transformers.ipynb#scrollTo=e7cqAWIayn6w

3.https://huggingface.co/blog/fine-tune-whisper

4.https://kyawkhaung.medium.com/multi-label-text-classification-with-bert-using-pytorch-47011a7313b9