Palestinian Speech Accent Recognition using Wav2Vec

Sary Hammad [1], *Laith AbuRob*[1]

[1] Department of Electric and Computer Engineering, Birzeit University, Birzeit, Palestine

`1192698@student.birzeit.edu, 1191024@student.birzeit.edu`

Abstract

This project focuses on the classification of Palestinian speech accents using the Wav2Vec2 model from the Hugging Face transformers library. A custom dataset was created from audio recordings, preprocessed to ensure consistency in length by concatenating shorter audio files to match a specified maximum duration. The Wav2Vec2 processor and sequence classification model were employed to convert raw waveforms into feature representations and perform the classification task. Training and evaluation datasets were organized into separate directories, with DataLoaders facilitating efficient data handling. The model was trained using specific hyperparameters, including mixed precision training and gradient accumulation to simulate larger batch sizes. Accuracy was used as the primary metric for model evaluation. The project aimed to leverage advanced machine learning techniques to accurately identify different Palestinian accents, providing a valuable tool for linguistic research and potential applications in speech recognition systems tailored to regional dialects. Initial results indicate promising performance, demonstrating the model's capability in distinguishing between distinct accents within the Palestinian context.

Index Terms: speech recognition, wav2vec, Palestinian accent classification

1. Introduction

The classification of speech accents represents a challenge in speech recognition and natural language processing. This project aims to classify Palestinian speech accents using deep learning techniques. The dataset is limited, comprising only 10 training samples per class across 4 distinct classes, necessitating advanced feature extraction methods to maximize data utility.

We employ the Wav2Vec2 model for feature extraction, leveraging its ability to process raw audio waveforms and extract meaningful features automatically. This model's transformer-based architecture enables it to capture intricate patterns in the speech data, crucial for accurate accent classification.

Our approach involves several key steps: data collection, preprocessing, feature extraction, model training, and evaluation. Preprocessing ensures all audio samples have a uniform length by concatenating shorter files. The Wav2Vec2 processor converts these waveforms into feature representations for the classification model. Training is optimized with specific hyperparameters to enhance performance despite the small dataset.

This project aims to enhance speech recognition systems' inclusivity and accuracy for regional accents, specifically focusing on Palestinian speech. The results provide valuable insights for future research and applications in speech processing and linguistics.

2. Background/Related Work

Many researchers have looked into similar topics, and we researched more than 30+ papers on similar topics.

2.1. Previous Research Papers

Accent classification has been a focal point of research in automatic speech recognition (ASR) due to its significant impact on the accuracy and robustness of speech recognition systems. Early studies, such as those by DeMarco and Cox, employed i-vector frameworks combined with Linear Discriminant Analysis (LDA), Subspace Discriminant Analysis (SDA), and Neural Classification Analysis (NCA) to enhance the recognition of native accents, achieving notable improvements in classification performance.[1]

A substantial body of work has explored the use of Mel-Frequency Cepstral Coefficients (MFCCs) for accent recognition. For instance, Brown utilized the Y-ACCDIST metric along with Support Vector Machines (SVM) to classify accents, achieving a high accuracy rate.[2] Similarly, Najafian and Russell integrated i-vector frameworks with Deep Neural Networks (DNN-HMM) to capture the intricate patterns of accent variations, underscoring the effectiveness of deep learning approaches in this domain.[3]

Recent advancements have focused on combining different feature extraction techniques and leveraging deep learning models to further enhance accent recognition. In one study, a Convolutional Neural Network (CNN) was employed alongside spectrogram-based features, resulting in superior performance compared to traditional methods. This highlights the potential of deep learning architectures in capturing the nuanced variations in speech patterns associated with different accents.[4]

Another notable approach involved the use of a Mixture of Experts (MoE) model for multi-accent speech recognition. This method, proposed by Jain et al., addressed the challenges of handling multiple accents by creating a unified framework that learns both phonetic and accent variability jointly. Their model demonstrated significant improvements in accuracy, showcasing the efficacy of MoE in managing accent diversity without the overhead of multiple specialized models.[5]

Another paper explores the development of an automatic speech recognition (ASR) system for Arabic, addressing the complexity introduced by multiple dialects and the scarcity of large, well-annotated corpora. The study introduces a large multi-dialectal corpus and a framework for training an acoustic model using deep neural networks, achieving state-of-the-art performance with a 14% error rate. The architecture combines convolutional and recurrent layers, processing spectrogram features of audio data, and aligns audio features with transcriptions using a beam search decoder with a tetra-gram language model. [6]

2.2. Wav2Vec

The Wav2Vec model, introduced by Schneider et al. at Facebook AI Research, represents a significant advancement in unsupervised pre-training for speech recognition. The model's primary objective is to leverage large amounts of unlabeled audio data to improve the performance of acoustic models in scenarios where labeled data is scarce.

2.2.1 Wav2Vec Processor

The Wav2Vec processor consists of two main components: the encoder network and the context network. The encoder network is a five-layer convolutional neural network that takes raw audio as input and converts it into a low-frequency feature representation. This representation encodes approximately 30 ms of 16 kHz audio and is computed every 10 ms. The context network then processes these encoded features to produce contextualized representations by mixing multiple time-steps of the encoder output. This network consists of nine layers, each employing a kernel size of three and a stride of one, resulting in a total receptive field of about 210 ms.

The primary task during pre-training is a contrastive loss that distinguishes a true future audio sample from negative samples. This is achieved by optimizing the network to predict future samples from a given context, effectively learning a robust feature representation from the raw audio data. This approach allows the model to generalize well across different datasets and improve the performance of downstream speech recognition tasks.

2.2.2 Wav2Vec Classifier

After pre-training, the representations produced by the Wav2Vec processor are fed into an acoustic model for supervised training. In their experiments, the authors used the wav2letter++ toolkit for training and evaluation. The acoustic model typically consists of multiple layers of convolutions, followed by non-linear activation functions and dropout for regularization. For instance, in the TIMIT phoneme recognition task, a seven-layer convolutional model with PReLU nonlinearity was used, which demonstrated significant improvements in accuracy when utilizing Wav2Vec pre-trained features compared to traditional log-mel filterbank features.

The Wav2Vec model has shown substantial improvements in word error rate (WER) on various benchmarks, including the Wall Street Journal (WSJ) dataset, where it outperformed the best character-based speech recognition models while using significantly less labeled training data. This demonstrates the model's ability to effectively leverage unsupervised pre-training to enhance speech recognition performance in both resource-rich and resource-poor scenarios. [7]

3. Methodology (System Description)

This project aims to classify Palestinian speech accents using a deep learning approach. The system framework comprises several key stages: data preprocessing, dataset creation, model setup, and training.

3.1. Data Preprocessing

The preprocessing step involves splitting long audio files into smaller, manageable segments. The provided script split_wav_file reads WAV files from the input directories, splits each file into 20-second segments, and saves up to three segments per file in the specified output directories. This ensures that all audio files are of uniform length, facilitating consistent model input.

3.2. Dataset Creation

The AccentDataset class is designed to handle the audio data and corresponding labels. This class loads the WAV files, ensures each audio sample matches the maximum length by concatenating shorter files, and processes the audio into a format suitable for the model using the Wav2Vec2 processor.

3.3. Model Setup

The model uses the Wav2Vec2 processor and sequence classification model from Hugging Face's transformers library. The processor converts raw audio waveforms into feature representations, and the classification model is fine-tuned for the specific task of accent classification.

3.4. DataLoader Configuration

The training and testing datasets are loaded using the DataLoader class from PyTorch, which facilitates efficient data handling and batching. The batch_size is set to 4, meaning each batch will contain 4 samples. The shuffle parameter is set to True for the training DataLoader to ensure that the data is randomized, which helps in better generalization by preventing the model from learning the order of the data.

3.5. Training Arguments

The training configuration is defined using the TrainingArguments class from the transformers library. Key parameters include:

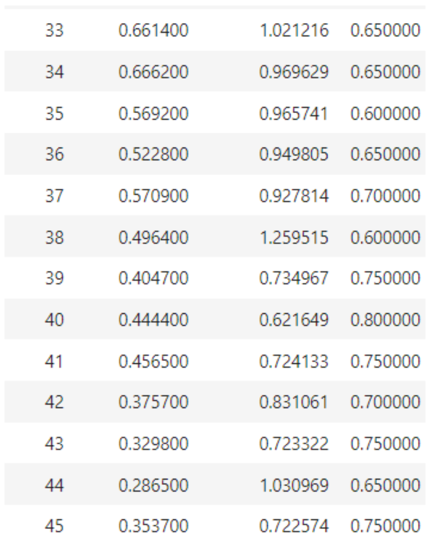
- output_dir: Directory where the model checkpoints and other outputs will be saved.
- evaluation_strategy: Set to "epoch" to evaluate the model at the end of each epoch.
- save_strategy: Also set to "epoch" to save the model at the end of each epoch.
- save_total_limit: Limits the number of saved checkpoints to 1 to save disk space.
- learning_rate: Set to 2e-5, which controls the step size during gradient descent.
- per_device_train_batch_size and per_device_eval_batch_size: Both set to 4 to match the DataLoader configuration.
- num_train_epochs: Set to 100, specifying the number of times the model will iterate over the entire training dataset.
- weight_decay: Set to 0.01, used to prevent overfitting by penalizing large weights.
- logging_dir: Directory where training logs will be stored.
- logging_steps: Logs training metrics every 10 steps.
- report_to: Set to "none" to disable reporting to external services.

- load_best_model_at_end: Ensures the best model based on evaluation metrics is loaded at the end of training.
- metric_for_best_model: Uses "accuracy" to determine the best model.
- fp16: Enables mixed precision training for faster and more efficient computation.
- gradient_accumulation_steps: Set to 4, which accumulates gradients over multiple batches to simulate a larger batch size.

4. Experiments and Results

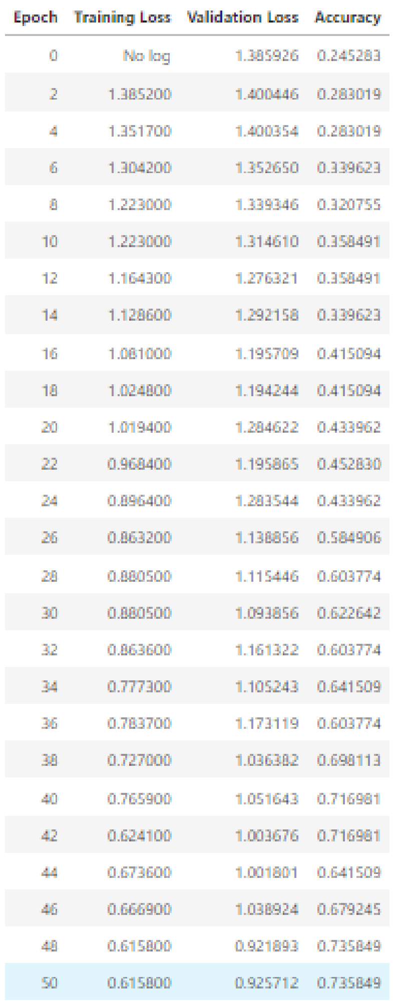
In the beginning, multiple simple models like Random Forests and SVM were trained and tested on the dataset but showed significant overfitting as the training accuracy was almost perfect 99%, yet the testing accuracy was around 25%, indicating that the model overfit the training data.

In addition, some CNN architectures from papers read were implemented, as well as HuBERT, yet they have shown significant underfitting and were not able to reach above 30% accuracy figures.



33	0.661400	1.021216	0.650000
34	0.666200	0.969629	0.650000
35	0.569200	0.965741	0.600000
36	0.522800	0.949805	0.650000
37	0.570900	0.927814	0.700000
38	0.496400	1.259515	0.600000
39	0.404700	0.734967	0.750000
40	0.444400	0.621649	0.800000
41	0.456500	0.724133	0.750000
42	0.375700	0.831061	0.700000
43	0.329800	0.723322	0.750000
44	0.286500	1.030969	0.650000
45	0.353700	0.722574	0.750000

The Wav2Vec model was first trained and tested on the first 10s of each wav file, and it reached a maximum test accuracy of 55%, which is more than double that of a random guessing model (25%). Then, we trained and tested the model on the first 20s of each wav file, the maximum accuracy achieved through this method was 80%. However, this meant that the model was not trained on all the dataset we have, and for it to be more robust, the final stage was splitting each wav file to windows of non-repeating 20s, and a maximum sample size of 3 per wav file (due to computational limits). This final training achieved an accuracy of ~73.5%, which is pretty impressive given the limited computational power and small dataset.



Epoch	Training Loss	Validation Loss	Accuracy
0	No log	1.385926	0.245283
2	1.385200	1.400446	0.283019
4	1.351700	1.400354	0.283019
6	1.304200	1.352650	0.339623
8	1.223000	1.339346	0.320755
10	1.223000	1.314610	0.358491
12	1.164300	1.276321	0.358491
14	1.128600	1.292158	0.339623
16	1.081000	1.195709	0.415094
18	1.024800	1.194244	0.415094
20	1.019400	1.284622	0.433962
22	0.968400	1.195865	0.452830
24	0.896400	1.283544	0.433962
26	0.863200	1.138856	0.584906
28	0.880500	1.115446	0.603774
30	0.880500	1.093856	0.622642
32	0.863600	1.161322	0.603774
34	0.777300	1.105243	0.641509
36	0.783700	1.173119	0.603774
38	0.727000	1.036382	0.698113
40	0.765900	1.051643	0.716981
42	0.624100	1.003676	0.716981
44	0.673600	1.001801	0.641509
46	0.666900	1.038924	0.679245
48	0.615800	0.921893	0.735849
50	0.615800	0.925712	0.735849

5. Conclusion

The project aimed to classify Palestinian speech accents using various machine learning and deep learning models, encountering significant challenges and making notable progress. Initially, simpler models like Random Forests and Support Vector Machines (SVM) were employed. These models demonstrated severe overfitting, with training accuracies nearing 99% but testing accuracies plummeting to around 25%. This stark contrast indicated the models' inability to generalize beyond the training data.

Subsequent attempts involved implementing Convolutional Neural Network (CNN) architectures inspired by recent research, as well as the HuBERT model. Unfortunately, these approaches resulted in significant underfitting, with accuracies failing to surpass the 30% mark.

The Wav2Vec model, however, marked a turning point. Initial training and testing on the first 10 seconds of each audio file yielded a test accuracy of 55%, which was more than double the baseline of random guessing (25%). Extending the training to the first 20 seconds of each file further improved accuracy to an impressive 80%. Yet, to ensure the model's robustness and comprehensive use of the dataset, the final strategy involved splitting each WAV file into non-repeating 20-second windows, with a maximum of three samples per file due to computational constraints. This approach achieved a final accuracy of approximately 73.5%.

Despite the limited computational resources and small dataset, the results obtained using the Wav2Vec model are remarkable. The significant improvement in accuracy underscores the potential of advanced deep learning models in handling complex tasks such as accent classification. Future work could explore leveraging larger datasets and more powerful computational resources to further enhance the model's performance and robustness.

In conclusion, this project explored various machine learning and deep learning approaches to classify Palestinian speech accents. Simpler models like Random Forests and SVM exhibited severe overfitting, while CNN architectures and HuBERT showed significant underfitting. The Wav2Vec model demonstrated substantial promise, with training on different durations of audio achieving accuracies up to 80%. The final approach of splitting each WAV file into non-repeating 20-second windows achieved a commendable accuracy of approximately 73.5%, given the computational constraints and limited dataset. Future work will aim to build on these findings, enhancing dataset size, preprocessing techniques, and model optimization to further improve performance and robustness.

6. References

[1] Jain, A., Singh, V. P., & Rath, S. P. (2019). A multi-accent acoustic model using mixture of experts for speech recognition. Interspeech 2019. https://doi.org/10.21437/interspeech.2019-1667

[2] Cetin, O. (2022). Accent recognition using a spectrogram image feature-based Convolutional Neural Network. Arabian Journal for Science and Engineering, 48(2), 1973–1990. https://doi.org/10.1007/s13369-022-07086-9

[3] Ghafoor, K. J., Hama Rawf, K. M., Abdulrahman, A. O., & Taher, S. H. (2021). Kurdish dialect recognition using 1D CNN. ARO-THE SCIENTIFIC JOURNAL OF KOYA UNIVERSITY, 9(2), 10–14. https://doi.org/10.14500/aro.10837

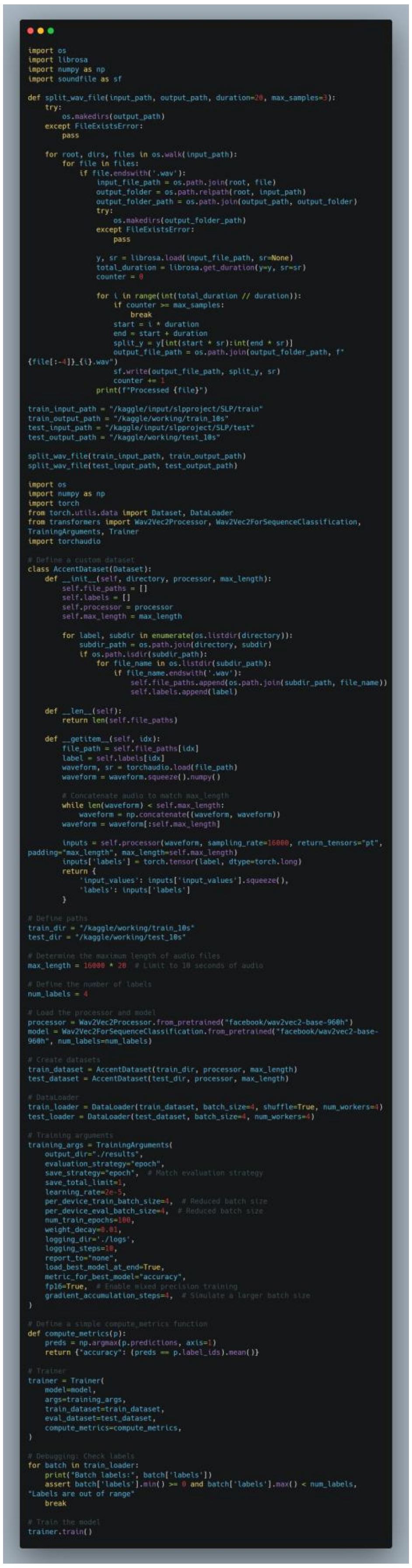
[4] Langari, S., Marvi, H., & Zahedi, M. (2020). Efficient speech emotion recognition using modified feature extraction. Informatics in Medicine Unlocked, 20, 100424. https://doi.org/10.1016/j.imu.2020.100424

[5] Salau, A. O., Olowoyo, T. D., & Akinola, S. O. (2020). Accent classification of the three major Nigerian Indigenous Languages Using 1D CNN LSTM network model. Advances in Computational Intelligence Techniques, 1–16. https://doi.org/10.1007/978-981-15-2620-6_1

[6] Ali, A. R. (2020). Multi-dialect Arabic speech recognition. 2020 International Joint Conference on Neural Networks (IJCNN). https://doi.org/10.1109/ijcnn48605.2020.9206658

[7] Schneider, S., Baevski, A., Collobert, R., & Auli, M. (2019). Wav2vec: Unsupervised pre-training for speech recognition. Interspeech 2019. https://doi.org/10.21437/interspeech.2019-1873

7. Code Appendix



```python
import os
import librosa
import numpy as np
import soundfile as sf

def split_wav_file(input_path, output_path, duration=20, max_samples=3):
    try:
        os.makedirs(output_path)
    except FileExistsError:
        pass

    for root, dirs, files in os.walk(input_path):
        for file in files:
            if file.endswith('.wav'):
                input_file_path = os.path.join(root, file)
                output_folder = os.path.relpath(root, input_path)
                output_folder_path = os.path.join(output_path, output_folder)
                try:
                    os.makedirs(output_folder_path)
                except FileExistsError:
                    pass

                y, sr = librosa.load(input_file_path, sr=None)
                total_duration = librosa.get_duration(y=y, sr=sr)
                counter = 0

                for i in range(int(total_duration // duration)):
                    if counter >= max_samples:
                        break
                    start = i * duration
                    end = start + duration
                    split_y = y[int(start * sr):int(end * sr)]
                    output_file_path = os.path.join(output_folder_path, f"
{file[:-4]}_{i}.wav")
                    sf.write(output_file_path, split_y, sr)
                    counter += 1
                print(f"Processed {file}")

train_input_path = "/kaggle/input/slpproject/SLP/train"
train_output_path = "/kaggle/working/train_10s"
test_input_path = "/kaggle/input/slpproject/SLP/test"
test_output_path = "/kaggle/working/test_10s"

split_wav_file(train_input_path, train_output_path)
split_wav_file(test_input_path, test_output_path)

import os
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import Wav2Vec2Processor, Wav2Vec2ForSequenceClassification,
TrainingArguments, Trainer
import torchaudio

# Define a custom dataset
class AccentDataset(Dataset):
    def __init__(self, directory, processor, max_length):
        self.file_paths = []
        self.labels = []
        self.processor = processor
        self.max_length = max_length

        for label, subdir in enumerate(os.listdir(directory)):
            subdir_path = os.path.join(directory, subdir)
            if os.path.isdir(subdir_path):
                for file_name in os.listdir(subdir_path):
                    if file_name.endswith('.wav'):
                        self.file_paths.append(os.path.join(subdir_path, file_name))
                        self.labels.append(label)

    def __len__(self):
        return len(self.file_paths)

    def __getitem__(self, idx):
        file_path = self.file_paths[idx]
        label = self.labels[idx]
        waveform, sr = torchaudio.load(file_path)
        waveform = waveform.squeeze().numpy()

        # Concatenate audio to match max_length
        while len(waveform) < self.max_length:
            waveform = np.concatenate((waveform, waveform))
        waveform = waveform[:self.max_length]

        inputs = self.processor(waveform, sampling_rate=16000, return_tensors="pt",
padding="max_length", max_length=self.max_length)
        inputs['labels'] = torch.tensor(label, dtype=torch.long)
        return {
            'input_values': inputs['input_values'].squeeze(),
            'labels': inputs['labels']
        }

# Define paths
train_dir = "/kaggle/working/train_10s"
test_dir = "/kaggle/working/test_10s"

# Determine the maximum length of audio files
max_length = 16000 * 20  # Limit to 10 seconds of audio

# Define the number of labels
num_labels = 4

# Load the processor and model
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base-960h")
model = Wav2Vec2ForSequenceClassification.from_pretrained("facebook/wav2vec2-base-
960h", num_labels=num_labels)

# Create datasets
train_dataset = AccentDataset(train_dir, processor, max_length)
test_dataset = AccentDataset(test_dir, processor, max_length)

# DataLoader
train_loader = DataLoader(train_dataset, batch_size=4, shuffle=True, num_workers=4)
test_loader = DataLoader(test_dataset, batch_size=4, num_workers=4)

# Training arguments
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",  # Match evaluation strategy
    save_total_limit=1,
    learning_rate=2e-5,
    per_device_train_batch_size=4,  # Reduced batch size
    per_device_eval_batch_size=4,  # Reduced batch size
    num_train_epochs=100,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=10,
    report_to="none",
    load_best_model_at_end=True,
    metric_for_best_model="accuracy",
    fp16=True,  # Enable mixed precision training
    gradient_accumulation_steps=4,  # Simulate a larger batch size
)

# Define a simple compute_metrics function
def compute_metrics(p):
    preds = np.argmax(p.predictions, axis=1)
    return {"accuracy": (preds == p.label_ids).mean()}

# Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=test_dataset,
    compute_metrics=compute_metrics,
)

# Debugging: Check labels
for batch in train_loader:
    print("Batch labels:", batch['labels'])
    assert batch['labels'].min() >= 0 and batch['labels'].max() < num_labels,
"Labels are out of range"
    break

# Train the model
trainer.train()
```