Speaker-Independent Spoken Digit Recognition (xSDR)

Anas Mohammed Ali
MSc Embedded Systems
anmo00003@uni-saarland.de
Saarland University

Meenu Anil
MSc Embedded Systems
mean00001@uni-saarland.de
Saarland University

Mrinal Mahindran
MSc DSAI
mrmr00001@uni-saarland.de
Saarland University

Abstract

This project explores the application of machine learning-powered automatic speech recognition (ASR) to the problem of spoken digit recognition (SDR). The goal is to develop a Speaker-Independent SDR system that can classify short audio clips of digits (0-9) and predict the digit spoken. This approach is efficient for identifying unique identifiers such as bank information, social security numbers, and postcodes. The system is designed to generalize to different speakers and languages with limited annotated speech data. The project tackles SDR as a sequence classification task using deep neural network models.

1 Introduction

The ability of a machine learning algorithm to recognize spoken digits (for example, 0 to 9) spoken by any individual, independent of accent or other speech characteristics, is known as "Speaker-Independent Spoken Digit Recognition (xSDR)".

The goal of this project is to create a Speaker-Independent Spoken Digit Recognition (xSDR) system that is powered by machine learning. The fact that different speech samples may have varying durations due to inherent speech variability is a problem with using spectrograms as a representation of speech features. We used a downsampling technique to acquire a fixed-size representation for all speech samples in order to address this. We then used the speaker-based train/dev/test segments to create a linear model using data from four speakers. We examined the model's performance using a confusion matrix of all potential labels and used accuracy as an evaluation measure (0-9).

The following task was to implement neural models that can take the entire speech segment as input without downsampling, in order to increase the precision of the machine learning-powered Spoken Digit Recognition (SDR) system. Recurrent Neural Networks (RNN) and Temporal Convolution Networks (TCN) are two different neural model classes that we investigated. These models were put into practice, and their effectiveness was compared to that of the baseline linear model learned on a downsampled signal representation of the speech segment. We also examined how the various models differentiate the various groups in the final non-linear layer using the dimensionality reduction algorithm t-SNE. We used a bootstrapping-based statistical significance test to ascertain whether the variations between the models are statistically significant. The baseline model was considered for this analysis.

We investigated how training machine learning models on speech data from a single speaker affected the model's performance for the third portion of the task. Then we augmented the data using two techniques, namely "Frequency Masking" and "Pitch Shifting". We also created views for the training data and used "Contrastive loss" based on these views.

2 Dataset

For this project, we use the Speaker-Independent Spoken Digit Recognition (xSDR) dataset, which consists of short audio clips of spoken digits (0-9). The dataset has already been divided into training, development, and test sets. In order to replicate a speaker-independent setting, evaluation speakers are not included in the training set.

The dataset includes recordings made by diverse speakers, ensuring that the system can adapt to speakers other than those in the training data. This also makes it possible for the system to be used in situations where only a small amount of annotated voice data is provided by a single speaker.

The `SDR_metadata.tsv` file, which includes details about the audio files including speaker identification, audio file path, and digit identifier, was read to explore the dataset using Python's Pandas library.

3 Model Architecture

In this project, we investigated three models: a linear model, a temporal convolutional network (TCN), and an LSTM recurrent neural network for speech emotion recognition (RNN).

3.1 Linear Model

Downsampled spectrograms of speech fragments were used to train the linear model. The spectrogram was divided into N identically sized divides across the time axis, and mean pooling was applied to each split across the frequency axis to perform the downsampling. In order to train the linear model, the resulting downsampled spectrogram was reorganized as a vector. Using SGDClassifier from scikit-learn, the model was trained using data from four individuals ('Nicolas,' 'Theo,' 'Jackson,' and 'George').

3.2 RNN

The simple RNN (Recurrent Neural Network) model for sequence classification takes the input and consists of an RNN layer with `num_layers` layers and `hidden_size` number of hidden units, followed by a fully connected layer (`nn.Linear`) that maps the output of the RNN layer to the output `classes (num_classes)`. The forward method initializes the initial hidden state h0 of the RNN to zeros and applies the RNN layer to the input sequence x along with the initial hidden state h0 to obtain the output sequence out. Finally, the hidden state of the last time step is decoded using a fully connected layer and returned as the output. If `use_lastlayer` is True, then a `softmax` activation is applied on the output before returning.

3.3 TCN

The 1-D Temporal Convolutional Network (TCN) is a deep learning model for sequence data. A fully connected layer is placed after a number of 1D convolutional layers in the TCN architecture. As input arguments, the user can specify the kernel size, the dropout rate, and the number of channels in each convolutional layer. Convolutional layers are created using the provided hyperparameters by the `create_conv_layers()` method, then applied to the input data by the `forward()` method, followed by global max pooling and a fully connected layer to produce the output. The output is processed through a softmax activation function if `use_lastlayer` is set to True.

4 Task Implementation

4.1 Task 1

4.1.1 Methodology

In this task, we used Mel spectrograms as input to a linear model to perform speaker recognition. We downsampled the spectrograms by taking the mean of every N column, where N is a hyperparameter that we can adjust to balance between model complexity and performance. We used scikit-learn's `SGDClassifier` as our linear model, which uses stochastic gradient descent to optimize a linear function that maps the input spectrograms to a vector of speaker identities. We created two functions to process the data. The first method, `downsample_spectrogram`, returns the downsampled spectrogram after receiving a spectrogram and the downsampling factor N as inputs. The second function, `load_data`, extracts the Mel spectrograms using the librosa library and downsamples the spectrograms using the `downsample_spectrogram` function after reading in a TSV file that maps audio file paths to speaker IDs. Additionally, the `load_data` function divides the incoming data into training, development, and test sets and normalizes it by taking the mean and dividing it by the training set's standard deviation. Using the training data as inputs and their associated speaker identities as targets, we trained the linear model using scikit-learn's fit method. The performance of the model was then assessed using the predict technique on the development set. To determine the model's accuracy on the development set, we used the `accuracy_score` utility of the scikit-learn package.

4.1.2 Training

Our code creates a linear model using Scikit-learn's SGDClassifier with a logistic loss function, a maximum of 1000 iterations, and a random state of 42. The model is trained on the downsampled and padded training samples using the `fit()` method. The accuracy, confusion matrix, and classification report of the model are evaluated on the dev and test splits using the `predict()` method and the `accuracy_score()`, `confusion_matrix()`, and `classification_report()` functions from Scikit-learn's metrics module. The evaluation results are printed to the console, including the dev and test accuracies, confusion matrices, and classification reports. The code aims to assess the performance of the linear model in classifying the down-

sampled and padded input data into 10 classes (0-9). The classification report provides metrics such as precision, recall, and F1 score for each class, along with their weighted average.

4.1.3 Evaluation

The model was evaluated on both development and test sets, and the accuracy, confusion matrix, and classification report were generated for each. The confusion matrix shows the number of true positives, false positives, true negatives, and false negatives for each class in the classification task. The diagonal elements in the matrix represent the correctly classified instances, and the off-diagonal elements represent misclassifications. The classification report shows precision, recall, and F1-score for each class, as well as macro-averaged and weighted averages across all classes. The accuracy of the model on the development set is 0.376, and on the test set is 0.419. The precision, recall and F1-score values vary for each class, indicating that the model performs differently in each class.

4.2 Task 2

4.2.1 Methodology

The methodology involves using a dataset to train and test two neural network models: a temporal convolutional network (TCN) and a recurrent neural network (RNN). Presumably, a training set, a validation set, and a test set has been created from the information. Following a completely connected layer and a hyperbolic tangent activation function, the TCN model has several convolutional layers with batch normalization, ReLU activation, and dropout. The RNN model comprises a fully connected layer, a hyperbolic tangent activation function, and an RNN layer with a predetermined number of hidden units and layers.

In order to find the ideal hyperparameters for each model, it also includes a hyperparameter search using either grid search or random search. The learning rate, number of epochs, number of layers, dropout rate, and concealed size for the RNN model are among the hyperparameters being looked for. The learning rate, the total number of epochs, and the failure rate are the hyperparameters for the TCN model that are being sought after. The models are assessed using the validation set to find the best hyperparameters. and converted the extracted features from each model's dimensionality to 2D using the t-SNE method. Plotted the reduced features using a scatter plot.

4.2.2 Training

The final models are trained using the merged training and validation sets and assessed on the test set after the best hyperparameters have been identified. The functions dev, test, and train are used to assess the models on the validation set and test set, respectively. The functions `train_model` are used to train the models.

After the best hyperparameters have been determined, the final models are trained using the combined training and validation sets and evaluated on the test set. The models are evaluated on the validation set and test set, respectively, using the methods dev, test, and train. The models are trained using the `train_model` tools.

4.2.3 Evaluation

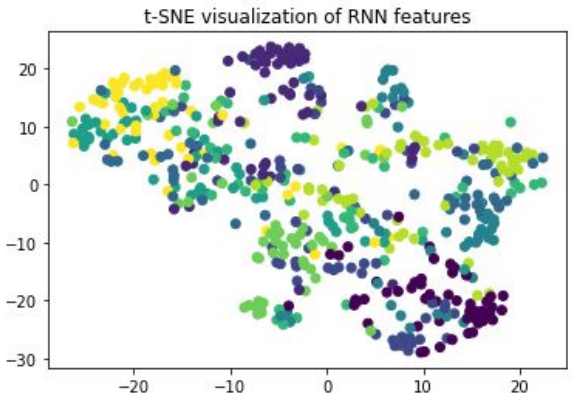

Figure 1: t-SNE visualisation of RNN features

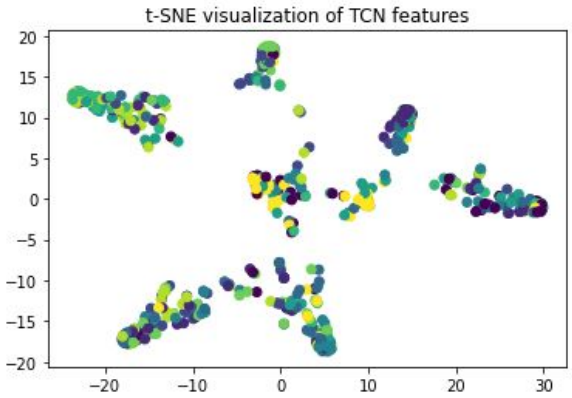

Figure 2: t-SNE visualisation of TCN features

We used accuracy as a measure to assess the performance of both our models. Our RNN model achieved a maximum accuracy of 45% on the development set. In order to assess the RNN model's generalization capabilities, we also evaluated it on the test set. The RNN model's accuracy on the test

set was 42%, showing that it is capable of generalizing to new data. Now in the case of our TCN model, we achieved a maximum accuracy of 41% and on the test set we got an accuracy of 43%. Figure 1 and 2 shows the resulting graph that we attained after using t-SNE.

4.3 Task 3

4.3.1 Methodology

In this task, we are given speech data from a single speaker (here, George), and we need to train our models on this subset of the data. However, since we want the model to generalize to any speaker, we need to use Data Augmentation techniques to artificially create more samples for each class. We can apply augmentations on the spectrogram (here, `SpecAugment`) and on the raw waveform before creating the spectrogram, such as `pitch manipulation`. We need to explore the effect of one type of augmentation from each type and report our observations while analysing the confusion matrices. Additionally, we can leverage speech data augmentation to create different "views" of each training sample in a stochastic or deterministic approach. To do this, we can implement at least one model using a contrastive loss based on different views of the training samples in a contrastive learning setting with a margin-based objective function. Finally, we need to compare the performance of the contrastive model with the one without contrastive learning and report and discuss our observations.

4.3.2 Training

Our models are trained on different training data. In the first part, we trained both models on a single speaker and assessed our models on the test set. For the second part, we applied frequency masking and pitch shift to the training data with a single speaker and then trained both the models on the modified dataset and assessed our models on the testing set. In the last part of the task, we created two different views of our training data using our frequency and pitch augmentations and used the contrastive loss to train the model on the different views of the data

4.3.3 Evaluation

From the evaluation metrics we can see that our models did not perform well on the testing data with a single speaker and the accuracy further decreased when we applied frequency masking and pitch shift to the training data. With frequency masking applied to our RNN model, we got an accuracy of 25% while our TCN model got an accuracy of 22%. Similar trends can be seen when we applied pitch shifting. For Single Speaker with data augmentation and Contrastive Loss, our models seemed to be struggling with several classes and gave really low accuracy values.

References

[1] Haider Al-Tahan and Yalda Mohsenzadeh, *CLAR: Contrastive Learning of Auditory Representations*, 2020.

[2] P. H. Le-Khac, G. Healy, and A. F. Smeaton, *Contrastive Representation Learning: A Framework and Review*,EEE Transactions on Pattern Analysis and Machine Intelligence, vol. 43, no. 12, pp. 4020-4038, 2021

[3] D. S. Park and W. Chan, *SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition*, 2019.

[4] Berg-Kirkpatrick, T., Burkett, D., and Klein, D, *An Empirical Investigation of Statistical Significance in NLP*,In Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL) (pp. 995-1005). Association for Computational Linguistics.