Speaker-Independent Spoken Digit Recognition (xSDR)

NNTI - Project
Suraj Sudhakar - 7015702
Ganavi Basavaraju - 7015432
Keerthana Krishnamurthy Burly - 7023672

Abstract

Spoken Digit Recognition (SDR) is a popular Audio Speech Recognition (ASR) task, and a variety of models exist for this task. In this report, we evaluate and compare the performance of different existing models. The effect of data shortage on the model performance and the effectiveness of data augmentation to remedy the issue are also investigated. The baseline model is created using a linear model and downsampled Mel spectrogram. The choice of DNN models are CNNs, recurrent neural networks like RNN, and LSTM models which work well for time series data. Self-supervised learning is also used to tackle the shortage of labeled data. We evaluate and compare our models using evaluation metrics like precision, f1, and recall and visualize using the Confusion matrix and t-SNE. The results for different models, scenarios, and methods are tabulated and discussed.

Index Terms — SDR, xSDR, ASR, RNN, LSTM, CNN, Machine learning, DNN

1 Introduction

Many modern applications, such as virtual assistants, and dictation systems, rely heavily on automatic speech recognition (ASR). Spoken digit Recognition (SDR) is a particular utilization of ASR where the objective is to recognize the expressed digits in a short brief snippet. SDR has practical applications in banking, and security, among other fields.

Neural networks (NN) have revolutionized the field of Machine Learning, and as such, have been used to solve the task of SDR. Due to a large number of models, it becomes crucial to be able to evaluate and compare them to choose the one suited for our task.

Consider a scenario where recognizing spoken digits from speakers other than those in the training set is required. A model working well on a dataset of speakers might not generalize well on unseen data. Therefore, the model should be evaluated against subtasks like Speaker-independent spoken digit recognition (xSDR).

Mel spectrogram, a matrix representation of speech data in the frequency domain is used to feed as input to neural network(NN) models. We establish a linear model as our baseline and use LSTM, and CNN as the main evaluation models. Initially, we evaluate the model performance on the complete training dataset which measures the overall capacity of the model to learn SDR classification.

To test against xSDR, we use only a portion of our data, from one speaker, for training. Popular solution for dealing with data shortage is data augmentation. The train data from one user is augmented and the model is evaluated against both scenarios, with and without augmentation. Evaluating against the task of xSDR gives more insight into model generalization.

The choice of evaluation metric plays a role in the overall evaluation procedure. We have used the accuracy metrics, precision, and F1 score. Our metric allows us to choose a model, for instance, if one is interested in precision over accuracy. Finally, we visualize the Confusion Matrix and t-SNE which give further insight into which label the model works better for and is important for model reasoning.

The above-described scenarios are conducted as our experiments and results are tabulated and discussed.

2 Data Exploration and Preprocessing

The spoken digits are from 0-9, our dataset consists of 3000 audio samples from 6 different speakers with 500 samples for each speaker. The dataset is divided into train, dev, and test sets. The training set has 2000 samples from 4 speakers, dev set: 497 samples, and the test set: 503 samples from 2 speakers. The number of classes is almost equal in each set.

The raw data is read and converted to Mel spectrogram, which yields a matrix of the form $\mathbf{X} \in R^T \times R^k$. The size of the matrix depends on the duration of the signal(T) and number of frequency bands(k). The frequency band (k) is fixed as 13, chosen from the recommended size list, for all our experiments so as to allow faster computation.

3 Experiment Models

3.1 Baseline Model

For the baseline model, SGD Classifier and Logistic Regression models are used. Spectrogram for different speech samples have different sizes, but linear classifiers expect fixed dimension input, therefore spectrogram is downsampled.

Interpolation is used as a downsampling technique because it can be interpreted as a form of convolution with a small window. We interpolate across the time axis for each frequency channel so that signal of $\mathbf{X} \in R^T \times R^k$ is converted to $\mathbf{X} \in R^N \times R^k$ where N is now a constant chosen by us. We chose N=25 as it suited the length of existing samples well. We had to perform padding when required. Finally, they are converted to $\mathbf{v} \in R^{Nk}$ vectors.

The linear classifiers (SGD and Logistic Regression) were trained on complete train dataset and validated and evaluated on dev and test datasets. The models used hinge and squared hinge loss respectively.

3.2 Deep NN Models

For deep NN architectures, RNN, LSTM, and temporal 1D CNN models are considered.

DNN models are capable of accepting data of varying lengths after tuning them. Therefore we use the original $\mathbf{X} \in R^T \times R^k$ mel spectrogram as our input. But due to varying input sizes, the batch size is set to 1. Custom dataset and dataloader were built for our purpose.

The RNN and LSTM architectures are used in this project because they are well-suited for processing sequential data such as audio signals. RNNs and LSTMs have the ability to maintain a memory of previous inputs, which can be useful in processing sequences of varying lengths.

The weights for all the models are initialized with a normal distribution with a mean of 0 and standard deviation of 1e-3, while the bias is initialized with zeros.

For the 'RNN architecture', a one-layer RNN with a hidden size of 128 units is chosen. It has a simple architecture that can be trained quickly and is very effective in processing sequential data. Since tanh activation function is demonstrated to be effective in these types of models, it is used. The model has 2 layers and is trained to classify into 10 classes.

For the 'LSTM architecture', a two-layer LSTM with a hidden size of 128 was chosen. LSTMs are a more complex type of RNN and is a better choice for long-term dependencies in sequential data. The output of the LSTM layer is passed through a linear (fully connected) layer which maps the hidden state vector to the output class labels of size 10. The initial hidden and cell states of the LSTM layer are set to zero, and for classification, the last hidden state output of the LSTM layer step is used.

The last architecture is 'Temporal 1D CNN'. It is a popular architecture used for sequence modeling tasks such as speech recognition, and natural language processing. The model consists of a series of 1D convolutional layers which execute convolution for each time channel, hence the name temporal.

The model takes Mel spectrogram as input with k frequency bins. The first layer of the model is a 1D convolutional layer with 32 filters, each of size 3 and with padding 1. Therefore for input of size $T \times k$, the output is $T \times 32$. This layer is followed by 3 more 1D convolutional layers, where each layer has 64, 128, and 1024 filters and takes as input the output from the previous layer.

Finally, we have an output of size $T \times 1024$. To convert it into a vector, we use a global max pooling function taking inspiration from the aggregate function used in Graph Neural Network (also found in Point cloud architectures). This operation selects the maximum value for each filter across the temporal dimension of the output, resulting in a fixed-length feature vector (1024). The feature vector is then passed through two fully connected linear layers, with ReLU activations, to make a prediction for the class label.

The choice of the CNN architecture for this project is motivated by the ability of CNNs to learn local, shift-invariant patterns in sequential data. The use of multiple convolutional layers allows the model to learn more abstract representations of the given data. The max pooling operation ensures that the model is invariant to the temporal length of the input and reduces the dimensionality of the feature vectors. Finally, the two linear layers provide a non-linear transformation of the feature vector to make the final classification.

3.3 Data Augmentation

One of the experiment is to use the train data from only one speaker and test model generalization on test and dev data sets to emulate insufficient data. To evaluate such scenarios, the temporal 1D CNN model is used since it got better results in earlier tasks. This model performed well on the training set, achieving high accuracy throughout the training. However, its performance on the validation set was more variable, with lower accuracy for some epochs. Test accuracy was lower compared to train accuracy, suggesting that the model may be overfitting. Hence for such tasks, data Augmentation is highly recommended.

SpecAugmentation is one of the widely used augmentation techniques for speech data. It technique combines time warping, frequency masking, and time masking. The spectrogram's time axis, which mimics variations in speech speed, is distorted by time warping. A continuous frequency section of the spectrogram is masked with frequency masking, and time masking hides a random portion of the spectrogram's time axis. All these 3 augmentation types are implemented and applied to SpecAugmentation and it can be seen in figure 1. It improves robustness and generalization in the models.

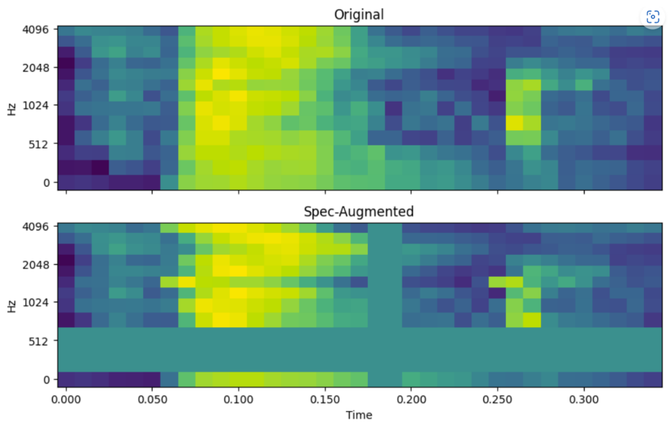

Figure 1: Orginal and SpecAugmented Spectrogram

Another technique is to augment the raw waveform data. Here, we use jittering and scaling, which are based on adding Gaussian noise to the audio signals as seen in Figure2. The reason for using this technique is to simulate real-world scenarios where the input audio signal may vary due to factors such as noise, and channel distortion.

These modified signals are used to train the 1D temporal CNN model to obtain better results.

	Training Accuracy	Validation Accuracy	Test Accuracy
Baseline - SGD	91.10%	50.50%	54.67%
LSTM	90.70%	52.11%	57.85%
Temporal 1D CNN	97.00%	62.58%	64.81%

Table 1: Accuracy for training on full data

	Average precision	Average recall	Average f1-score
Baseline - SGD	0.616	0.543	0.538
LSTM	0.582	0.576	0.555
Temporal 1D CNN	0.656	0.640	0.612

Table 2: Metrics for training on full data

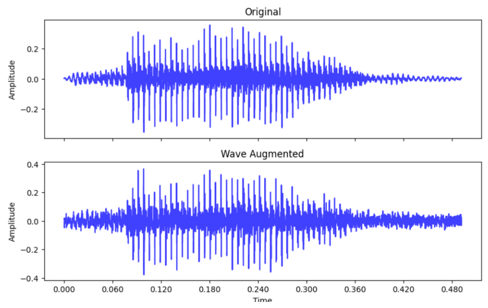

Figure 2: Orginal signal and Augmented signal

4 Experiment Results

4.1 Training on Complete dataset

As discussed earlier, we initially trained our models on the complete dataset to test which model achieve the highest accuracy, precision, F1 score. The train set is the complete 2000 samples in this case. The accuracy metrics for baseline, LSTM and temporal 1D CNN are provided in the table 1

Generally, we observe a significant difference in training accuracy versus test accuracy for all the 3 models, suggesting that the models are overfitting. The accuracy is generally higher for DNN models. The precision, recall, and F1 scores for the model on the test set are show in table 2

The metrics for every class are averaged and tabulated. We have the best results for Temporal 1D CNN, so we only show the Confusion Matrix for that in figure 3. Analyzing the Confusion Matrix, we see that no particular label is overly preferred or ignored. This means that model is not favorably predicting a certain label, but variation in precision per class is present.

In order to select the right hyper-parameters for our best model, we ran it for different learning rates and regularization parameter values. The training accuracy is visualized as shown in figure 4. Based on the curve obtained, we used a learning rate of 1e-3. A similar evaluation was carried out for the regularization parameter and the value of 1e-3 was found to work best.

4.2 Model analysis using t-SNE

Dimensionality reduction techniques like t-SNE and UMAP are used to visualize the model performance. Again, we only show the best CNN model in figure 5, the remaining can be seen in the notebook for comparison between the different models used. Similar labels are clustered together in t-SNE mapping, the model has learned to group similar things together and hence can classify well.

4.3 Training on Partial data

We train on the partial train set, where we chose data from one speaker. Only our baseline and CNN models were trained and evaluated in this experiment. Later, the CNN model was trained with

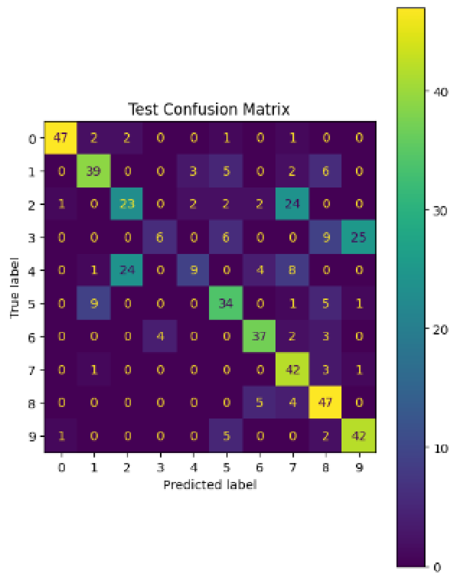

Figure 3: Confusion matrix - CNN Classifier on test set

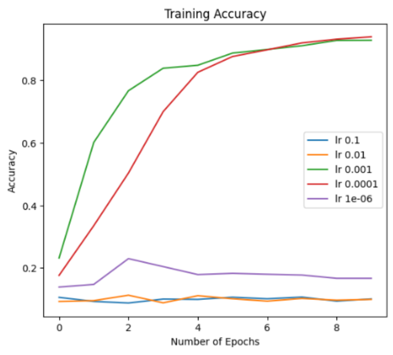

Figure 4: Train Accuracy Vs Different Learning rates

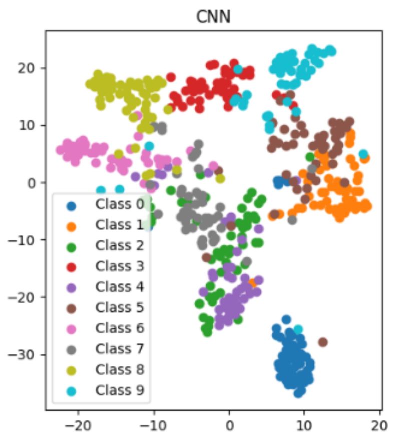

Figure 5: t-SNE for CNN

the augmented data and evaluated. The results are tabulated in the table 3

Looking at the results obtained, the data augmentation used was inadequate as we get similar accuracy on the test data, with or without augmentation. One of the reasons could the model we are using is comparatively small, and the other being the amount of data is very low. But looking at the confusion matrix for data with and without augmentation as shown in the figure 6,

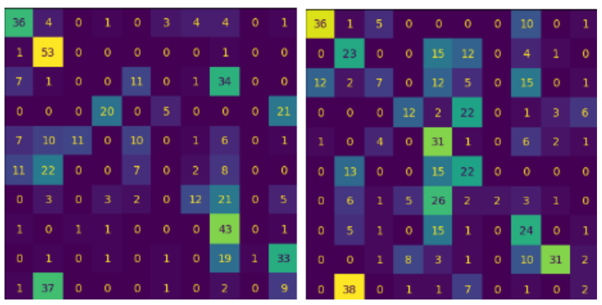

Figure 6: Confusion matrix without data augmentation (left) Vs Confusion matrix with spec augmentation (right)

We can see that the correct predictions are not

	Training Accuracy	Validation Accuracy	Test Accuracy
Baseline - SGD w/o data augmentation	97.60%	35.21%	37.77%
Temporal 1D CNN w/o data augmentation	96.80%	38.03%	36.58%
Temporal 1D CNN with spec augmentation	90.40%	35.83%	37.77%
Temporal 1D CNN with wave augmentation	100.00%	27.16%	29.62%

Table 3: Accuracy for training on partial data

dominated by a few labels. The correct predictions are more evenly spread out across classes which is more desirable.

4.4 Contrastive learning

The complete dataset without any labels was used to train a contrastive learning model based on SimCLR(Chen et al., 2020). The data and model was provided by us, model was changed to a simple CNN encoder, similar to temporal 1D CNN. This is a self-supervised method without any labels using a contrastive loss function. Model was trained on complete data, but due to not using labels, the accuracy obtained can be treated as testing accuracy.

Contrastive Learning Accuracy	84.38%

The data augmentation was very effective in the self-supervised setting. The high accuracy obtained makes it a desirable approach.

5 Conclusion

The choice of models for existing tasks is high, therefore it is essential to be able to analyze and evaluate the model in different scenarios to choose a model which works best for our purpose. The choice of models, hyperparameters, and evaluation metrics all are important in deciding the final model. For the task of SDR, we considered a baseline linear model, LSTM, and CNN. After evaluating the models, we can conclude that DNN models work better in general. The difference in RNN and CNN approaches was found to be statistically insignificant for the task at hand.

DNN models tend to overfit when the amount of training data available is less. The remedy of data augmentation was found to be not universal. The choice of which data augmentation, the amount of data, and the method being used are equally important. Though the amount of data available plays a crucial role in the final accuracy for the supervised setting, we saw that was not the case for the self-supervised scenario suggesting that where one method fails, another method could perform well with the same data and model.

6 References

References

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR.