Speaker-Independent Spoken Digit Recognition (xSDR)

Soham Roy and **Somrita Ghosh** and **Balaji Venkatesan**

1 Abstract

The objective of this project was to develop and compare various neural network models for speech recognition of digits, and to explore the impact of data augmentation on model accuracy. The models used for the task included CNN, RNN, LSTM, and contrastive learning. To assess the performance of the different models, a test set of spoken digit recordings was used. The models were trained on a training set and their accuracy was evaluated on the testing set. The results showed that data augmentation was effective in improving the accuracy of the models. In summary, this project highlights the effectiveness of deep learning models for speech recognition of digits and emphasizes the significance of data augmentation in enhancing model accuracy.

2 Introduction

We're developing a Spoken Digit Recognition system using deep neural network models to classify short audio clips of digits 0-9. Generalization to unseen data is crucial, especially for different speakers and accents. We use training, development, and test splits to evaluate and prevent overfitting. The training set trains model parameters, the development set fine-tunes hyperparameters, and the test set evaluates performance on completely new data. This split helps prevent bias and ensures the model generalizes well.

3 Implementation of Tasks

3.1 Task 1

3.1.1 Data Exploration

The data preprocessing task described involves reducing the number of timeframes in a given Mel spectrogram to a specified dimension N. The strategy being used is to insert frames from the resultant timeframe in equidistant timeframes with the previous timeframe value and bring the value to N. For example, if the resultant timeframe has 43 frames and N is 25, the timeframe value is converted to the next integer divisible by N, which is 50 in this case. Then, 7 frames are sampled from the resultant 43 frames in equidistant frames (frames 0, 6, 12, etc.) and inserted in the respective blocks to get an N of 50. Finally, a min sampling is done to get a final dimension of 25 in this particular example case

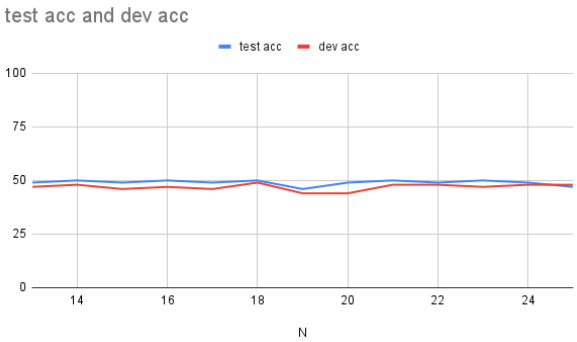

Figure 1: Test accuracy and Dev accuracy

3.1.2 Model

To prepare the data for training a linear model, we standardized it using the StandardScaler function from the sklearn.preprocessing module. We then utilized the SGDClassifier algorithm from the sci-kit-learn module to train the model. Log-loss was chosen as the loss function, with a regularization term of alpha = 0.001 and a ratio of L1 to L2 regularization of 0.35. The elastic net (Zou and Hastie, 2005) method was used for regularization. We fine-tuned the hyperparameters by testing different ranges of values and selecting the set of hyperparameters that produced the highest validation accuracy.

3.1.3 Summarization

After the data preprocessing, a simple logistic regression with L2 loss is used to evaluate accuracies for a range of N values from 13 to 25. The accuracy

values for both the dev and test datasets are hovering between 47 and 51. For precision and recall, the overall trend is that the higher recall,the lower precision, and vice versa, except for certain values such as 0, 1, and 9. The assumption is that the Mel spectrogram data for these values is very different from the others, allowing the logistic regression model to identify them with considerable accuracy and a better F1 score for those values. [Refer to Figure 1]

3.2 Task 2

3.2.1 Data Exploration

We have converted the audio signals into Mel spectrograms and have stored the resulting images in the dev, test, and train datasets, based on their respective sources. Since the length of the spectrogram image is dependent on the length of the input signal, it may vary.

3.2.2 Model 1: CNN:RESNET 50

Architecture

ResNet-50 consists of 50 layers, including convolutional layers, pooling layers, fully connected layers, and shortcut connections. The network uses skip connections or shortcut connections to allow information to flow directly from earlier layers to later layers without passing through intermediate layers. This helps to alleviate the problem of vanishing gradients and allows the network to be trained to very deep levels.

Pretrained Models

The basic steps to use a pre-trained model involve loading the model, preparing the input data, passing the data through the model to obtain predictions, and optionally fine-tuning the model. Using a pre-trained model can be beneficial as it saves time and resources by leveraging the knowledge and expertise of the pre-trained model, which was trained on a large and diverse dataset.

Training

We fine-tuned the last layer of ResNet-50, which is the classification layer, to output 10 class labels, utilizing transfer learning in the training process. Cross entropy was utilized as the loss function, and Adam optimizer was used as the optimization technique. The learning rate was set to 1e-5, and the learning rate decay was set to 0.99. The model was trained for 30 epochs.

3.2.3 Summarization

The summary of the training results in ResNet50 transfer learning is that the training accuracy reaches 98-99% in less than 30 epochs, while the test and validation accuracy is only 56%. The F1 score is 70% highest for certain classes like 7 and 1 . Analyzing the confusion matrix, it is evident that the model performed well for classes 7 and 1 with accuracy over 80%, but poorly for classes 5, 2, 3, and 4 with accuracy less than 25%.

3.2.4 Model 2: CNN: From Scratch(Musaev et al., 2019)

Architecture

CNN is a type of neural network commonly used in audio or image processing tasks. A typical CNN has several layers of convolutions and pooling operations which can extract the features from the input data. The pooling layer reduces the size of feature maps preserving the important features. The output of the Convolution layers is fed into one more fully connected layer, which performs classification or regression tasks on the extracted features. [For the visual representation refer to Figure 2 and Figure 3]

Training

We have used 3 layers of CNN and Relu activation function, and max pooling layers with Adam optimizer and learning rate of 0.001. We ran the model to a standard number of epochs(50).

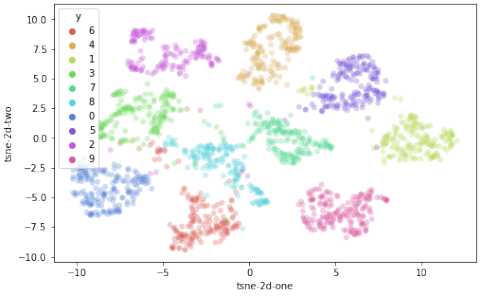

Figure 2: TSNE For CNN Train

3.2.5 Model 3: RNN

Architecture

Recurrent Neural Networks (RNNs) are a type of neural network that has a "memory" element that allows them to process sequential data.RNNs are commonly used for tasks such as speech recognition, language modeling, and time-series prediction.

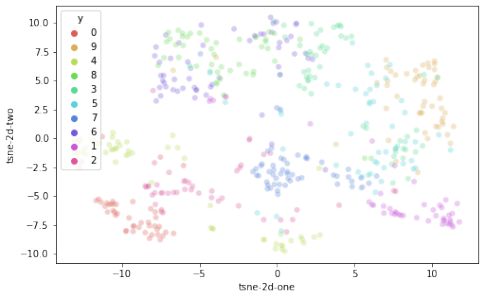

Figure 3: TSNE For CNN Test

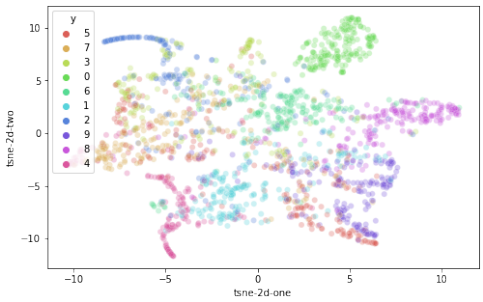

Figure 4: TSNE for RNN Training set

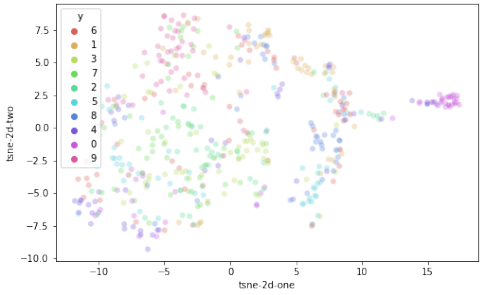

Figure 5: TSNE for RNN Testset

They can be implemented using various architectures, including simple RNNs, LSTM, and GRU.

Training

We have trained this model to the training data for 30 epochs, using batch training (Gunasekaran et al., 2021) to update the weights and biases at each step. The RNN with input dimension 13, hidden dimension 32, and output dimension 10 is used using stochastic gradient descent as the optimizer. we see after a few epochs the gradients are vanishing so the cross entropy loss becomes nan to overcome this we use gradient clipping to clip the gradients. [For the visual representation refer to Figure 2 and Figure 3] The F1score of older models are worse but in this it is between 35 and 70 and it implies that the model has enhanced its capacity to generalize to novel data.

Summarization

The RNN model has been trained for a classification task. The model has been trained for 40 epochs and has achieved an overall test and validation accuracy of 44 percent. However, the model is performing very well for classes 0 and 1, with F1 scores. On the other hand, the model is performing poorly for classes 3, 4, and 5, with F1 scores of less than 30 percent on average. The results obtained from using RNNs on the data are not conclusive, as shown by the t-SNE plot (Van der Maaten and Hinton, 2008) which does not clearly indicate cluster separation.

3.2.6 Model 4: LSTM - Single directional Architecture

(Hochreiter and Schmidhuber, 1997) LSTM is a type of RNN architecture that is designed to handle the vanishing gradient problem in traditional RNNs. It is particularly effective for processing sequential data, such as speech signals or time-series data.

Training

We have used LSTM with single directional with 30 epochs, and SGD optimizer with 13 input dimensions and 32 hidden dimensions.

3.2.7 Model 5: LSTM - Bi-directional

This model allows the network to take into account both past and future contexts when making predictions. Compared to the classical LSTM model, this one processes in both forward and backward directions, and the hidden states from both directions are concatenated to produce the final output. This allows the model to capture dependencies in both past and future contexts, which can be particularly useful in tasks like speech recognition.

Training

In the same way, as we have trained the LSTM with a single direction we used the same input parameters.

3.3 Task 3

3.3.1 Data Augmentation

Data augmentation is a technique used to increase the size of a training dataset by applying various

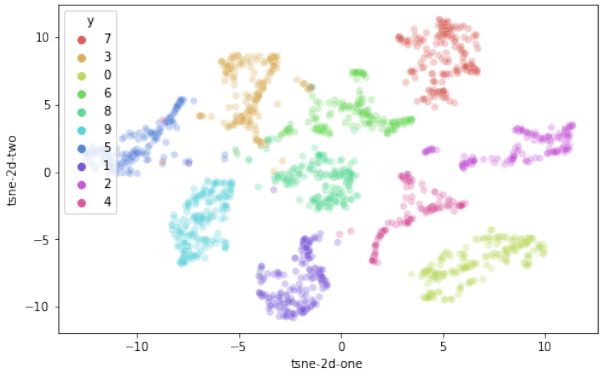

Figure 6: TSNE for LSTM

transformations to the existing data. It can help prevent overfitting by introducing more variety into the training data, which can improve the model's ability to generalize to new data.

3.3.2 DataPreprocessing

For this particular task, we employed a dataset and performed augmentation using two distinct techniques. Subsequently, we employed contrastive learning to train the model in order to enhance its overall accuracy of the model. We randomly chose the type of augmentation that we need to apply for a sample.

3.3.3 Model: Contrastive learning

(Koch et al., 2015)The contrastive learning model has been prepared using a Convolutional Neural Network (CNN) model as the backbone. The CNN model has been trained from scratch, and the output of the model in the penultimate layer is a 128-dimensional vector. The model output is a 10-dimensional vector.The 128-dimensional vector is being used to calculate the pairwise contrastive loss. $\mathrm{L}(i_1, i_2, Label) = \alpha \times (1 - Label) D_w^2$ $+ \beta \times Label \times [max(0, m - D_w)]^2$ This loss function is designed to bring similar classes closer together and push dissimilar classes apart. This is achieved by calculating the distance between the two vectors and using a margin to ensure that the distance between similar classes is smaller than the distance between dissimilar classes.In addition to the pairwise contrastive loss, the binary cross-entropy loss is being used as the overall loss function. This loss function is commonly used for binary classification tasks and is used to calculate the difference between the predicted and actual values.Overall, the contrastive learning model appears to be a promising approach for improving the performance of the CNN model. By using the pairwise contrastive loss, the model is better able to distinguish between similar and dissimilar classes, which can improve its accuracy and generalizability. Additionally, using the binary cross-entropy loss can further improve the model's ability to classify the data accurately.

Model Name	Test Accuracy
CNN with ResNet	56%
CNN from Scratch	57%
RNN	44%
LSTM	54%
Bi-directional LSTM	59.97%
Contrastive Learning	60%

Figure 7: Comparison of models based on accuracy

Conclusion

This project aimed to address the challenge of limited labeled data in classification tasks.[See Figure 7 for comparisons of the different models]. A baseline linear model was trained, followed by comparison of different Deep Neural Network architectures. However, these models showed good performance on training data but failed to generalize well. Contrastive learning was then explored to enhance the models' robustness. Due to RAM limitations, the training process was limited to a smaller number of epochs. A CNN-based architecture pretrained with contrastive learning and further trained for downsampling achieved the best results, with 60 percent accuracy on the testing data. This project highlights the effectiveness of contrastive learning for improving DNN model performance in classification tasks with limited labeled data.

References

Jashwant Raj Gunasekaran, Cyan Subhra Mishra, Prashanth Thinakaran, Mahmut Taylan Kandemir, and Chita R Das. 2021. Cocktail: Leveraging ensemble learning for optimized model serving in public cloud. *arXiv preprint arXiv:2106.05345*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. 2015. Siamese neural networks for one-shot

image recognition. In *ICML deep learning workshop*, volume 2. Lille.

Muhammadjon Musaev, Ilyos Khujayorov, and Mannon Ochilov. 2019. Image approach to speech recognition on cnn. In *Proceedings of the 2019 3rd International Symposium on Computer Science and Intelligent Control*, pages 1–6.

Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of machine learning research*, 9(11).

Hui Zou and Trevor Hastie. 2005. Regularization and variable selection via the elastic net. *Journal of the royal statistical society: series B (statistical methodology)*, 67(2):301–320.