Speaker-Independent Spoken Digit Recognition (xSDR)

Subrat Kishore Dutta
sudu00001@stud.uni-saarland.de

Prathvish Mithare
prmi00001@stud.uni-saarland.de

Abstract

Deep neural networks have become an important model for natural language processing applications. Deep networks often achieve their strong performance through supervised learning, which requires a large labeled dataset at a cost of human labor. When the data is limited learning the initial weights using Self-supervised learning approaches provides an effective way to improve a model's performance on the downsampled task. In this report, we summarize our work, which focuses on understanding and implementing Various Neural Network architectures for the task of Automated Speech Recognition. After a detailed statistical analysis for all the different models and learning approaches CNN based architecture pretrained with a Contrastive learning approach and further trained for the multi-class classification task achieves the best performance with a testing accuracy of 94 %.

1 Introduction

Machine learning-powered automatic speech recognition (ASR) systems have achieved remarkable progress in recent years, especially with the use of deep neural networks. This project aims to develop deep neural network models that can classify short audio clips of spoken digits accurately. This problem is treated as a classification task, where the input is an audio clip, and the output is the digit spoken in the clip. The developed system can be used in scenarios where there is limited annotated speech data from a single speaker, but the system needs to work on any speaker of that language.

2 Task Implementation

The purpose of this paper is to examine the execution of a task and evaluate the factors that play a significant role in determining its outcome.

2.1 Task 1

The task involved processing the incoming Mel-spectrogram of variable length into a form accepted by the linear model. This variability in the duration is due to the different inherent traits of the speeches by the speakers(1). The implementation has three steps. The first step involves downsampling each incoming Mel spectrogram to have the same fixed temporal resolution which is then followed by flattening them into one-dimensional vectors. These representations are then used to train the linear model. In the end, the trained model is evaluated on the testing and the validation set.

2.1.1 Preprocessing the Mel Spectrogram

As the Mel spectrogram with respect to each speech sample is different in its temporal resolution the study downsamples each of them into a fixed size length such that each frequency has the same sequence length. The fixed length is given by the variable N which is fed as a user-defined input to the downsampling function. The input length is padded with zeros such that the sequence length is divisible by N. Dividing the padded sequence with N gives the support region for the aggregating operation as shown in Figure 2. For the aggregation, we use absolute pooling which compares the absolute values of the minimum and maximum values of the slice and propagates the one with a larger magnitude. This is done so that the variations in the sequence can be better represented (2). Each sample obtained is now a 2D sequence where the K rows represent K frequencies and the N columns represent the temporal dimension. This is then flattened to form a single one-dimensional vector of length N*K.

2.1.2 The Linear Model

Before the Linear Model is trained the data is standardized using the StandardScaler function in the sklearn.preprocessing module. The standardized

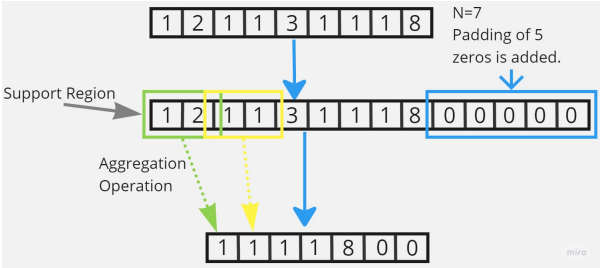

Figure 1: **Top:** Input Sequence, **Middle:** Padded Sequence, **Bottom:** Downsampled Sequence

data is then used to train the linear model, which was initiated using the SGDClassifier from the scikit-learn module(5). Log-loss was used as the loss function regularised with elasticnet (4) with a regularisation term, alpha of 0.001. The ratio of the L1 to L2 term is kept at 0.35. The hyperparameters finally used were tuned using the validation set, where different ranges of values were tested and then the one with the maximum validation accuracy is chosen.

2.1.3 Model Evaluation

The Model's performance was evaluated using the accuracy metrics on the dev and test split of the dataset. The confusion matrix of the model for the classification task is drawn from which precision, recall, and F1-score for each label were then analyzed.

2.2 Task 2

The study compared three models for processing sequential audio data: 1-D temporal convolution, Recurrent DNN with LSTM nodes(7), and GRU nodes(6). The models were optimized using the validation split of the dataset and trained until convergence using the training split. The same evaluation metrics were used to analyze the trained models as in the baseline case, and t-SNE (8) was used to compare how they separate different classes.

2.2.1 Preprocessing the Mel Spectrogram

The preprocessing involved to feed the data into the networks is different for the Convolutional Neural Network based architecture than that of the Recurrent Neural Network based architectures. The Mel spectrogram obtained, for each acoustic sample decomposes it into 13 frequencies with the sequence length varying between them. For the RNN-based models, the RNN layers accept the data in a particular format due to this the study took a transpose of the input sample dimension so that they turn in the form of sequence length x frequencies. As a result of this, each sample after taking a transpose has a dimension of Mx13, where M is the sequence length and 13 is the number of frequencies. The sequence length for each sample is kept as it is from the Mel spectrogram.

The CNN architecture contains pooling layers that reduce the dimension of the feature map by twice at each layer. In order to make the resultant dimension of the feature maps divisible by two the initial Mel Spectrogram was padded such that the length of the sequence is a power of two. The study pads the Mel spectrogram of each sample with zero resulting in each sample having a dimension of 13x256. In both cases, the data is normalized.

2.2.2 Model 1: Convolutional Deep Neural Networks

The Architecture The architecture has six convolutional blocks which are followed by two fully connected layers with 350 and 10 nodes respectively. The first fully connected layer is followed by a ReLU activation while the second one is followed by a softmax. Each convolutional block is composed of two convolutional layers with padding configured to "same" followed by a batch normalization layer, ReLU activation, and max pooling respectively. Since batch normalization follows the convolutional layers the bias term is set to zero while initializing the convolutional layer. Each convolutional layer is followed by a dropout layer. The number of kernels in the first three blocks increases successively after which it reduces in the next three blocks. A kernel size of 3 is used throughout the convolutional layers and a dropout probability of 0.2 is used which is obtained as the optimal value by testing on the validation set. The model has a total of 170260 trainable parameters.

The Training The validation set is used to find the optimal hyperparameter values like the regularization term, dropout probability, and the optimization algorithm. The set which gave minimum validation loss was selected for further training. The optimal value of λ, dropout probability, and optimizer turned out to be 0.005, 0.2, and Adam optimizer respectively. The training phase involved training the model for 1610 epochs out of which the initial 110 epochs were run with a learning rate of 0.001 followed by 500 epochs each of reduced learning rates of 0.0005,0.0001,0.00005 respectively. Training loss and the norm of the weight

gradient were monitored.

2.2.3 Model 2: Recurrent DNN with LSTM nodes

The Architecture The input to the network is first passed through a bidirectional LSTM layer which accepts an input size of 13 which essentially is the number of frequencies from the Mel spectrogram. The dimension of the output from each hidden state is kept at 300, and a total of 3 LSTM layers are stacked together. A dropout probability of 0.3 is used which is again obtained by evaluating a range of values on the validation set. The output of the LSTM layer is then fed through a series of four fully connected layers to successively reduce the dimensionality of the output and extract features. This also helps in adding more non-linearity to the model. The number of nodes in the first three successive levels is 200,100,50 respectively with the final output layer producing a 10-dimensional vector representing the classification probabilities of the input data. The model accounted for a total of 5231460 trainable parameters.

The Training Similar to the training of the CNN model the validation set is used to find the optimal hyperparameters. The optimal value of λ, dropout probability, and optimizer turned out to be 0.001, 0.3, and Adagrad optimizer respectively. The model was trained for 300 epochs out of which the initial 100 epochs were run with a learning rate of 0.001, followed by 200 epochs with a learning rate of 0.0005. Training loss and the norm of the weight gradient were monitored.

2.2.4 Model 2: Recurrent DNN with GRU nodes

The Architecture The architecture of this architecture is precisely similar to the previous architecture except for the change of the bidirectional LSTM nodes to GRU nodes. The model accounted for a total of 1452960 trainable parameters. Having the same structure, the GRU network had some clear advantages over the LSTM ones specifically as far as computational efficiency is concerned. GRU-based architectures have fewer gates and memory cells which makes them faster furthermore they have fewer parameters and hence are memory-efficient.

The Training The same setup for training is used where first the optimal values for the hyperparameter are determined and then the model was trained using the same. The optimal value of λ, dropout probability, and optimizer turned out to be 0.005, 0.3, and Adagrad optimizer respectively. Then 300 iterations of training followed, from which the first 100 epochs were trained with a learning rate of 0.001 followed by 200 epochs at 0.0005. Training loss and the norm of the weight gradient were monitored. A relatively faster convergence was observed while training this model.

2.2.5 Model Evaluation and Visualization

The same methodology for evaluation is followed as in task 1. In order to visualize how the classes are separated by the models t-SNE is used. The study also analyses if the difference between the models is statistically significant or not (9).

2.3 Task 3

The main goal of this task is to explore how we can train the model with less data and improve its ability to generalize. Based on previous tasks, it was determined that a CNN architecture is more effective for the given audio dataset. Therefore, in this task, we will continue to use the CNN architecture for analysis.

2.3.1 Training the model with fewer data

The model was initially trained using only audio files corresponding to 'George', resulting in a small dataset. Due to the limited amount of data, the model was unable to generalize effectively, leading to high test error rates.

2.3.2 Generalization

To generalize more, data augmentation (11) was used, where the data quantity is increased by using augmentation such as adding the noise, shifting time, shifting the pitch, and stretching the duration.

2.4 Contrastive learning

The task involves two stages where the model is pretrained in a Contrastive learning approach (3; 10) initially, and then the learned weights are used for the multi-class classification task. To this end, the study uses a similar CNN-based model used in the previous task. The CNN layers are kept the same and a few more linear layers are added to it so that an embedding of 128 dimensions can be created for each audio sample. Another fully connected layer with 10 nodes was used following which would predict the class of each sample. The model thus as a whole produces an embedding and a prediction for

the given sample. The prediction contributes to the cross-entropy loss and the embedding contributes to the Contrastive loss based on the sample pairs fed in the training loop(3). The dataset of positive and negative pairs is created by adding noise to the input sample of the same class and using a sample from a different class respectively. The model is then trained on this data based on the loss function given in (3) for 1500 epochs with a learning rate of 0.01 and Adam optimizer. The trained CNN layer is then used for the classification task.

3 Results and Discussion

The performance of each and every model on the testing set is summarised in Table 1. The linear model was used as a baseline so that the other models can be compared. The model achieved a training accuracy of 95.39 % but the validation accuracy achieved was very low with only 35.21 %. This reduction is mainly due to the simple nature of the linear model. The CNN-based model which achieved a training accuracy of 99.7 % improved significantly on the validation accuracy. After multiple attempts were made to add regularisation to the training so that it can be improved the best validation accuracy achieved was 67.60 %. Which when compared to its training accuracy clearly points towards overfitting. The RNN based models with bidirectional LSTM and GRU nodes showed similar training accuracy of 99.85% and 99.90 % however failed on the validation with a mere 52 % 55% validation accuracy. It is noteworthy that while evaluating on the development split to decide the hyperparameters, introducing dropout and higher regularisation in the layers and loss function respectively, significantly improved the validation score for all the models. Upon checking the statistical significance of the difference between these models it turned out to be significant. When analyzing the speech data of a single speaker (in this case, George), the quantity of data available to train the model is limited. Consequently, the model struggles to generalize effectively on other samples, leading to lower test accuracy of 44 %. Despite this, the model achieves a training accuracy of 98 %, indicating that the model overfits the data. In the subsequent experiments, various techniques for data augmentation have been examined, all of which are applied directly to the raw waveform. All of the models produced better test results, suggesting that they achieved better generalization after undergoing data augmentation. The results of the experiments are presented in 1. The model trained with the Contrastive learning approach however achieved the best results both in terms of validation and test accuracy with 93.36 % and 94.43% respectively. The result's optimality can be concluded to be contributed by the pretraining step involved which helps the model to learn general features first which results in better generalization when used on the downsampling task as the model is able to extract more generic information than fixating onto specific feature it learned with normal training.

Evaluation Metrics						
Model	Pr	Re	f1	Tr acc	Val acc	Test acc
Linear	0.62	0.40	0.41	0.95	0.35	0.40
CNN	0.71	0.71	0.70	0.99	0.68	0.70
LSTM	0.55	0.54	0.50	0.99	0.52	0.52
GRU	0.54	0.53	0.51	0.99	0.53	0.55
CNN(Si)	0.30	0.35	0.29	0.99	0.34	0.35
CNN(N)	0.44	0.45	0.44	0.99	0.45	0.45
CNN(St)	0.38	0.36	0.33	0.97	0.40	0.37
CNN(Sp)	0.45	0.47	0.43	0.95	0.46	0.45
CNN(Sd)	0.39	0.38	0.37	0.98	0.36	0.38
CNN(M)	0.52	0.46	0.44	0.96	0.48	0.48
CNN(C)	**0.96**	**0.96**	**0.96**	0.99	**0.93**	**0.94**

Table 1: Performance analysis of the various models based on various metrics. (Pr: Precision, Re: Recall, f1: f1-Score, CNN(Si): Trained with one speaker,CNN(N): Augmentation with noise, CNN(St): Augmentation with shifting time, CNN(Sp): Augmentation with shifting pitch, CNN(Sd): Augmentation with stretching duration, CNN(M): Mixing all Augmentation types, CNN(C): Trained with Contrastive learning)

4 Conclusion

In this project, we learnt mainly about scenarios where labeled data is limited and methods to overcome it. We started with training a linear model as the baseline and compared other Deep Neural Networks architecture for the classification task. All the models when trained directly for the classification task gave good training errors but failed to generalize. We found that using the Contrastive learning approach makes the model robust. The best results were achieved with the CNN-based architecture which was pretrained with contrastive learning and further trained for the downsampling task. It achieved a testing accuracy of 94.43 %.

References

[1] Z. -H. Ling et al., "Deep Learning for Acoustic Modeling in Parametric Speech Generation: A systematic review of existing techniques and future trends," in IEEE Signal Processing Magazine, vol. 32, no. 3, pp. 35-52, May 2015, doi: 10.1109/MSP.2014.2359987.

[2] Y. Sunaga, R. Natsuaki and A. Hirose, "Similar land-form discovery: Complex absolute-value max pooling in complex-valued convolutional neural networks in interferometric synthetic aperture radar," 2020 International Joint Conference on Neural Networks (IJCNN), Glasgow, UK, 2020, pp. 1-7, doi: 10.1109/IJCNN48605.2020.9207122.

[3] Mohsenzadeh, Y. (2020). CLAR: Contrastive Learning of Auditory Representations. ArXiv. /abs/2010.09542

[4] Zou, H., Hastie, T. (2005). Regularization and Variable Selection via the Elastic Net. Journal of the Royal Statistical Society. Series B (Statistical Methodology), 67(2), 301-320. https://doi.org/10.2307/3647580

[5] Lei Huang, Jie Qin, Yi Zhou, Fan Zhu, Li Liu, and Ling Shao. Normalization Techniques in Training DNNs: Methodology, Analysis and Application. arXiv preprint arXiv:2106.05345, 2021.

[6] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *CoRR*, abs/1412.3555, 2014. http://arxiv.org/abs/1412.3555.

[7] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667. URL https://doi.org/10.1162/neco.1997.9.8.1735.

[8] van der Maaten, Laurens and Hinton, Geoffrey. Visualizing Data using t-SNE. Journal of Machine Learning Research 2008

[9] Taylor Berg-Kirkpatrick, David Burkett, Dan Klein. An Empirical Investigation of Statistical Significance in NLP.

[10] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov. Siamese Neural Networks for One-shot Image Recognition.

[11] Tom Ko1, Vijayaditya Peddinti, Daniel Povey, Sanjeev Khudanpur Audio Augmentation for Speech Recognition.

5 Figures

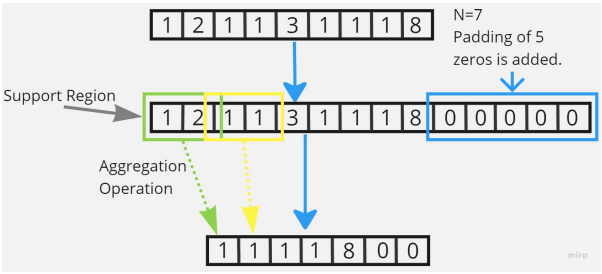

Figure 2: **Top:** Input Sequence, **Middle:** Padded Sequence, **Bottom:** Downsampled Sequence

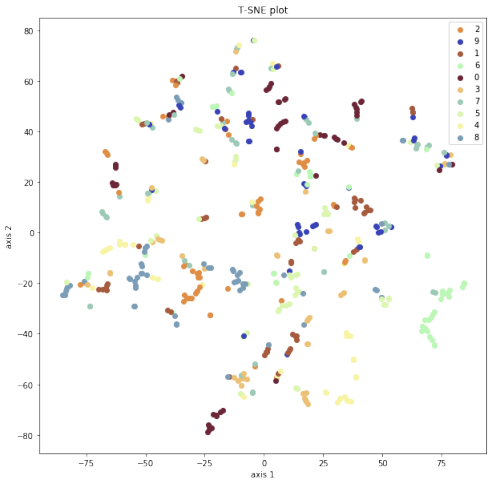

Figure 3: Visualization of the embeddings created by the Linear baseline model with their classes using t-SNE

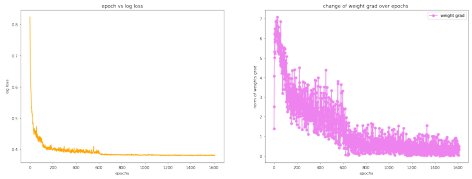

Figure 4: Training curve for CNN-based model

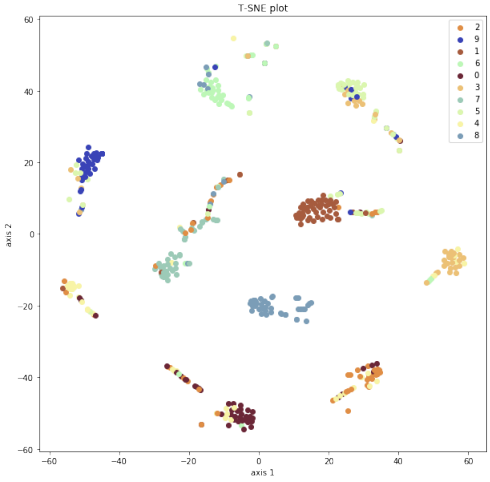

Figure 5: Visualization of the embeddings created by the CNN-based model with their classes using t-SNE

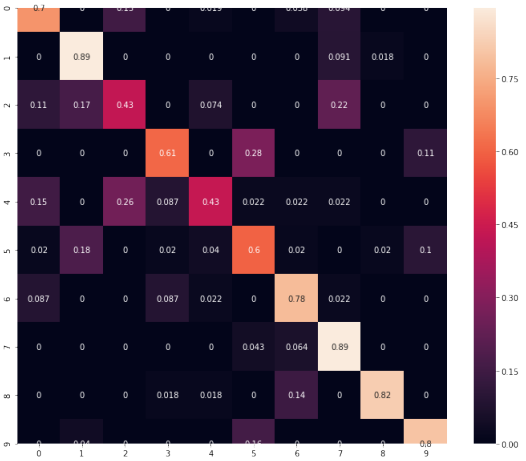

Figure 6: Confusion matrix for the performance of CNN-based model on test data

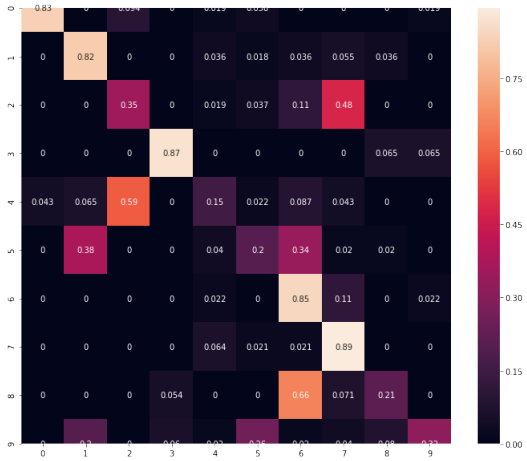

Figure 9: Confusion matrix for the performance of LSTM-based model on test data

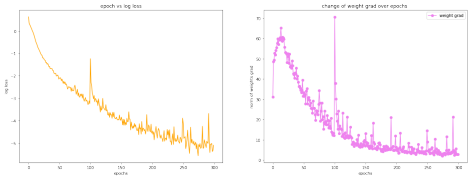

Figure 7: Training curve for LSTM-based model

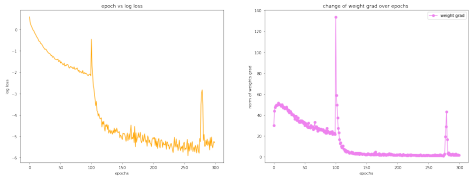

Figure 10: Training curve for GRU-based model

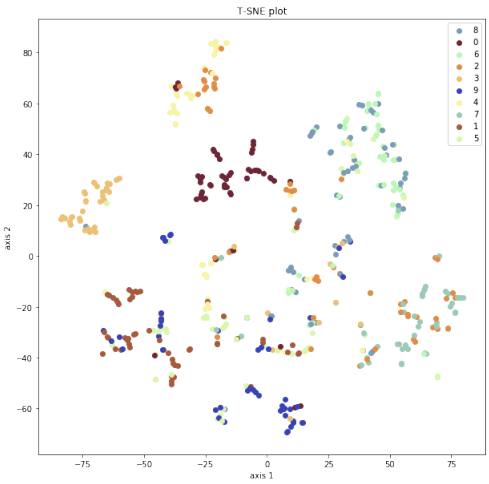

Figure 8: Visualization of the embeddings created by the LSTM-based model with their classes using t-SNE

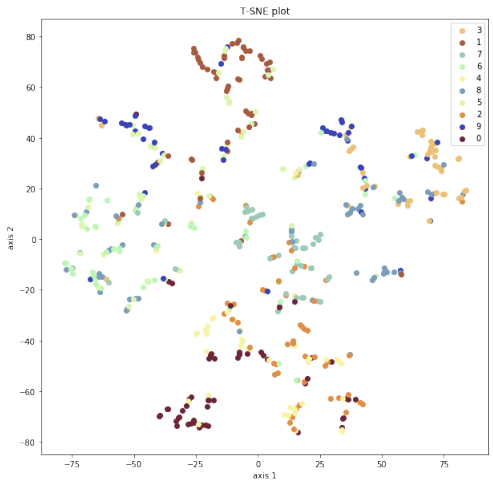

Figure 11: Visualization of the embeddings created by the GRU-based model with their classes using t-SNE

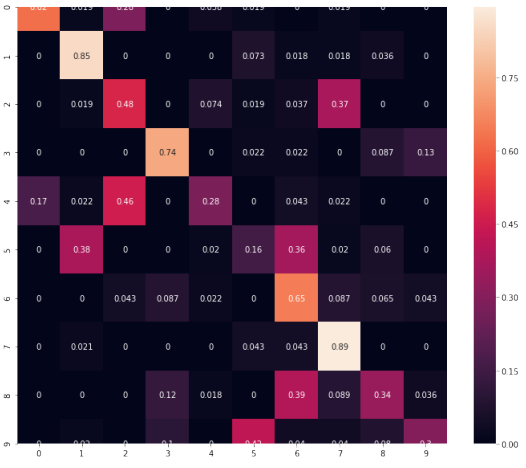

Figure 12: Confusion matrix for the performance of GRU-based model on test data

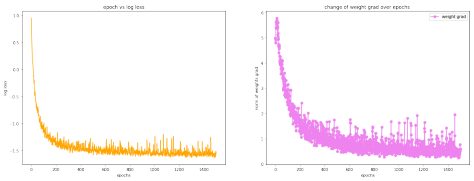

Figure 13: Training curve for CNN(Co) in Contrastive learning approach

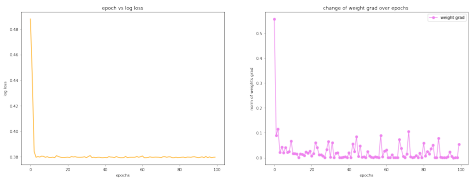

Figure 14: Training curve for CNN(Co) for the down-sampling task

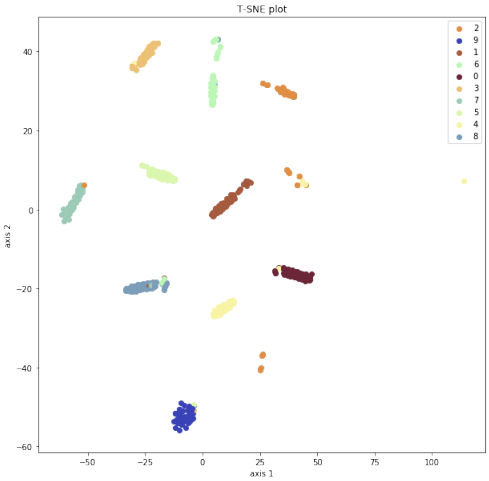

Figure 15: Visualization of the embeddings created by the CNN(Co) with their classes using t-SNE

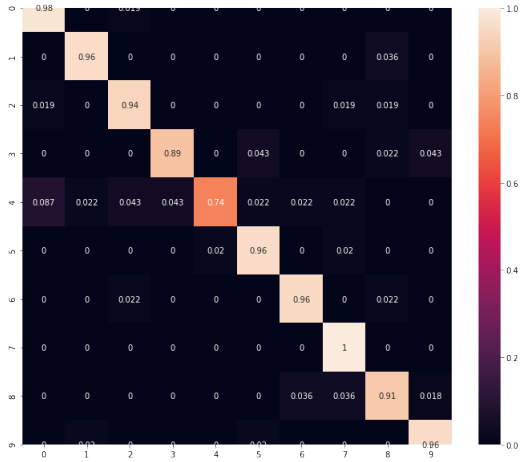

Figure 16: Confusion matrix for the performance of CNN(Co) on test data

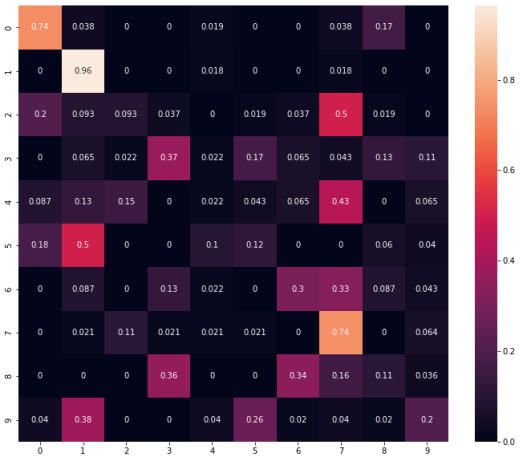

Figure 17: Confusion matrix for the performance of the model without data augmentation

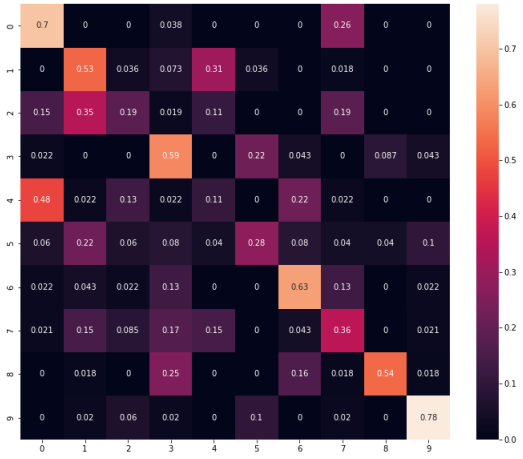

Figure 18: Confusion matrix for the performance of the model with noise augmentation

Figure 19: Confusion matrix for the performance of the model with time shifting augmentation

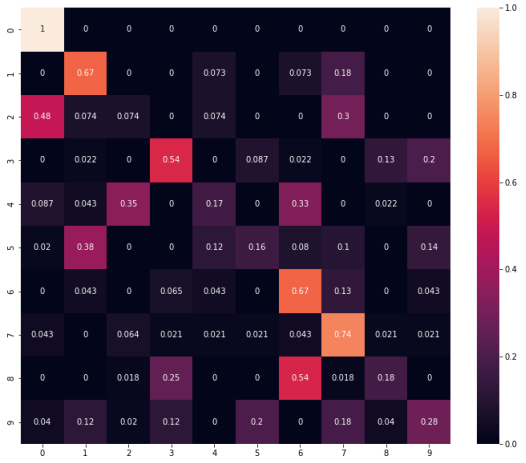

Figure 20: Confusion matrix for the performance of the model with pitch shifting augmentation

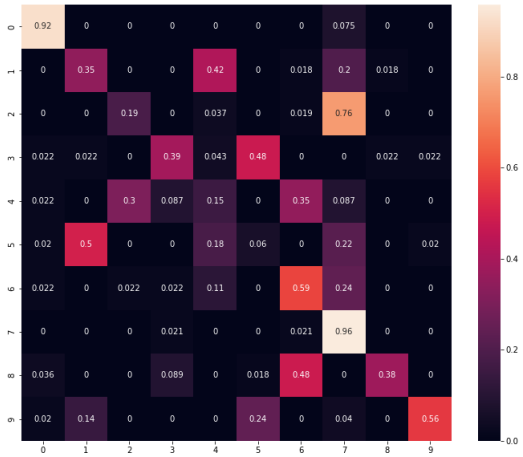

Figure 22: Confusion matrix for the performance of the model with a mix of all augmentation

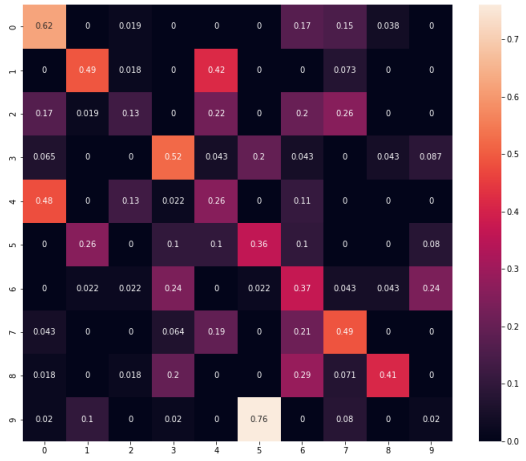

Figure 21: Confusion matrix for the performance of the model with duration stretching augmentation