Automatic Speaker Recognition Challenge

Jona te Lintelo (s1021036), Alma Nilsson (s1102888)

1 Introduction

This project report describes our approach to the given speaker recognition challenge as well as the experiments that led to our submission. The goal of this challenge is to build an Automatic Speaker Recognition system and make it perform optimally for speaker detection, given the conditions of this challenge.

2 Method

2.1 Networks

In order to create a well performing model for speaker recognition we attempted to implement some of the most recent types of neural networks used within the context of speaker recognition. Some recent approaches[11, 7, 5, 3, 2] use variations of ResNet[6]. The ResNet architecture aims to solve increasingly bad performance of deeper neural networks by introducing residual blocks. These residual blocks are used to allow gradients to flow through a network directly, without passing through non-linear activation functions.

We implemented three variations of the ResNet architecture, ResNet10, ResNet18 and a ResNeXt[9] network. The architectures can be seen in table 1. ResNeXt is an extension of the resnet architecture. The improved functionality ResNeXt provides is having more non-linearity between blocks, meaning a more parallel instead of a sequential system. It achieves this by adding a "cardinality" dimension to the system (normally a ResNet only has a "height" and "width" dimension) [10]. Our implementation of ResNet for this project has one-dimensional convolutions. This choice results in the addition of 1×1 convolution layers after every residual block. Whereas usually 1×1 convolution is used after a set of residual blocks. The prototype skeleton network with the default settings served as a baseline for our results. The ResNeXt network was used for the final submission on the leaderboard. For all networks the spectrogram is used as input to the network.

2.2 Hyperparameters and Optimization

Some important choices to be made during the implementation of the neural network are the optimization function, the hyperparameters for the optimization function and hyperparameter values for training. For this project we used the Adam optimizer and Negative Log-Likelihood loss because of their usage for optimization in recent related work[11, 7, 4]. The Adam hyperparameters that we explored are learning rate and learning rate decay. Learning rate determines the size of updates performed. Learning rate decay determines by how much the learning rate is decreased after a certain frequency of steps or epochs. Decaying the learning rate can mitigate overfitting, if the learning rate remains large we may simply end up bouncing around or out of a minimum and thus not reach optimality. The training hyperparameters that we experimented with are batch size, learning rate and the number of epochs.

The general effects of hyperparameters on our networks were determined by performing ablation studies on the learning rate and batch size for ResNet10 and ResNet18. We selected three different values for batch size and learning rate. The batch sizes selected were 64, 128 and 256. These values are based on related work which show a full-sized VoxCeleb dataset with ResNet34 and ResNet50 perform better on larger batch sizes, such as 256 and 512[7]. Additionally, these larger batch sizes can result in shorter training times and without the need of complex training schedules to reach similar performance[8]. Hence, this range of values seemed appropriate in relation to the literature, our smaller dataset and less complex ResNet implementations. The learning rates analysed were 0.001, 0.0006 and 0.0001. The motivation for these values is based on the related work and the assumption that more complex networks benefits from smaller learning rates than the default 0.003 due to the more complex optimization surface. All three values for batch size and learning rate were paired. Considering these nine runs as a pilot study enabled us to save time. By determining what the effects were of certain hyperparameter values we determined the best range of values for final testing optimization faster.

Table 1: Modified ResNet10 and ResNet18 architectures with a mean statistical pooling layer at the end. Batch normalisation is used before the rectified linear unit (ReLU) activations. Each row specifies the number of convolutional filters and their sizes as size × size, # filters. Square brackets indicate blocks over which there are residual connections. The input to each network is the spectrogram. 1×1 convolution for ResNeXt is only used after every first residual block in a layer and uses Lazy layers[10].

Layer Name	ResNet10	ResNet18	ResNeXt
conv1	7×7, 32, stride 2		7×7, 128, stride 2
pool1	3×3 MaxPool, stride 2		
conv2	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \\ 1\times1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \\ 1\times1, 64 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \\ 3\times3, 256 \end{bmatrix} \times 2$
conv3	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \\ 1\times1, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \\ 1\times1, 128 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \\ 3\times3, 512 \end{bmatrix} \times 2$
conv4		$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \\ 1\times1, 256 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 1024 \\ 3\times3, 1024 \\ 3\times3, 1024 \end{bmatrix} \times 2$
conv5		$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \\ 1\times1, 512 \end{bmatrix} \times 2$	$\begin{bmatrix} 3\times3, 2048 \\ 3\times3, 2048 \\ 3\times3, 2048 \end{bmatrix} \times 2$
pool2			3, AdaptiveAvgPool
linear1			128-d FC
pool3	128, MeanStatPool1D	512, MeanStatPool	128, MeanStatPool
prediction	100, LogSoftmax		

2.3 Data augmentation

To see if we could increase the performance of the networks, we decided to augment the data. The original training data contains 100 unedited audio files with a 50/50 division of female and male voices. First off, noise was injected into the data, in order to reduce overfitting and make the model more robust to varying types of input [1]. Random noise was generated for each separate file with a constant noise factor of 0.01. Secondly, a random speed change of the data was induced resulting in longer sound waves, in essence "stretching" out the time. This can make it easier to recognize certain patterns in sound waves. Thirdly, random gain was applied, this randomly changes the loudness of the audio data, to account for a varying intensity of input.Furthermore, reverberation was implemented using a one-dimensional convolutional layer. Reverberation is the reflection of sound in a room (against walls or the ceiling), which can also be described as echo. Since echo is often encountered in audio data, it is useful to train your system on reverberated audio. All reverberation was done with a room impulse response ranging from 1.01 to 1.3 of that of the original data. Lastly, we applied pitch change. It changes the pitch randomly, and in the resulting data the sound peaks are a bit less high and softer sounds get amplified. An illustration of each augmentation, their impact and the degree of change, can be seen in Figure 2 in the appendix.

These augmentations were implemented both online and offline. The 'online' implementation was done as part of decoding of the individual *wav* files in the data processing python file *datapipe.py*, where each sample was implemented an augmenting method by random probabilities. We also implemented an offline locally stored library to increase efficiency, with a separate copy for each of the types of data augmentation, thus increasing the dataset to five times the original size. So, in the second case, the data augmentation expanded the dataset with new labeling, effectively allowing the network to train on a larger dataset. No grid search, or equivalent analysis, to determine the importance of the effect of a single augmentation on the accuracy of the model was done.

3 Results

Results of our ablation study are shown in Figures 3 through 6. Additionally, the results of experiments with data augmentation are given in Figures 7 through 8. Lastly, an overview is given of the results of the leaderboard submission. Our baseline concerns the prototype network. The baseline yields testing equal error rate(EER) of 27.34 at learning rate = 0.003 and batch size = 128.

3.1 Hyperparameter pilot study

With ResNet10, all combinations of batch sizes and learning rate converge at a similar accuracy range of around 0.4, with the exception of batch size = 256 and learning rate = 0.0001. The validation loss shows overfitting for all combinations of batch size and learning rate = 0.001 or learning rate = 0.0006. The larger learning rates reach a minimum loss in a range slightly above 2.5 for all batch sizes. After this point the loss becomes unstable and starts to overfit. A smaller learning rate of 0.0001 for all batch sizes show that the loss stops overfitting and is more stable. Although, with a smaller learning rate it takes more steps to reach a minimum loss at around 2.4. All graphs show some form of plateauing for the accuracy. For ResNet18, similar observations are made as was for ResNet10. Notable differences are that the maximum accuracy is lower at a range of 0.25-0.35 and minimum loss reached is higher in a range around 3.0 instead of slightly above 2.5.

3.2 Data augmentations

ResNet10 and ResNet18 were trained for 75 epochs with batch size = 256, learning rate = 0.00015 and a decay rate $\gamma = 0.95$ applied after every epoch. ResNeXt was trained for 30 epochs with, learning rate = 0.002, batch size = 64 and a decay rate $\gamma = 0.8$ for every 10 epochs. The results for each model, with and without data augmentation, can be seen in Table 2. Figures 7 and 8 show the validation accuracy and validation loss of each of the models when training with and without the augmented data. The two ResNet models are seen to somewhat plateau in terms of accuracy and both have stable validation loss. Resnext, on the other hand, does not show signs of plateauing and has an increasingly unstable loss, especially in the case of the model trained on the augmented data.

In the case of ResNet10 and ResNeXt we see a decrease in the testing error for the validation data of 0.74 and 0.11 respectively, whereas in the case of ResNet18 there was a increase in error of 0.19 when the model was trained with augmented data.

Table 2: Testing EER for ResNet10. ResNet18 and ResNeXt, with and without data-augmentation.

Model	Without data-augmentation	With data-augmentation
ResNet10	23.05 %	22.41 %
ResNet18	23.22 %	23.41 %
ResNeXt	22.05 %	**21.96 %**

3.3 Final submission

The final optimization was performed with the ResNeXt architecture described in Table 1. The ResNeXt architecture was trained for 100 epochs with learning rate = 0.002, $\gamma = 0.8$ and batch size = 64. The best testing EER achieved was 19.03. This results was obtained by training on the non-augmented dataset. From Figure 1 it can be seen that our validation loss in the final submission is overfitting. After 10 to 15 epochs the minimum loss is reached. The validation accuracy shows convergence around 0.55.

4 Discussion

Overall, the results indicate that our implementations of ResNet and ResNeXt perform worse when compared to recent work on VoxCeleb speaker recognition. Nevertheless, there are several interesting influences of the hyperparameters on performance in the pilot studies. Additionally, our results show that there is still room for improvement in terms of optimization of hyperparameters and data augmentation for our implemented networks.

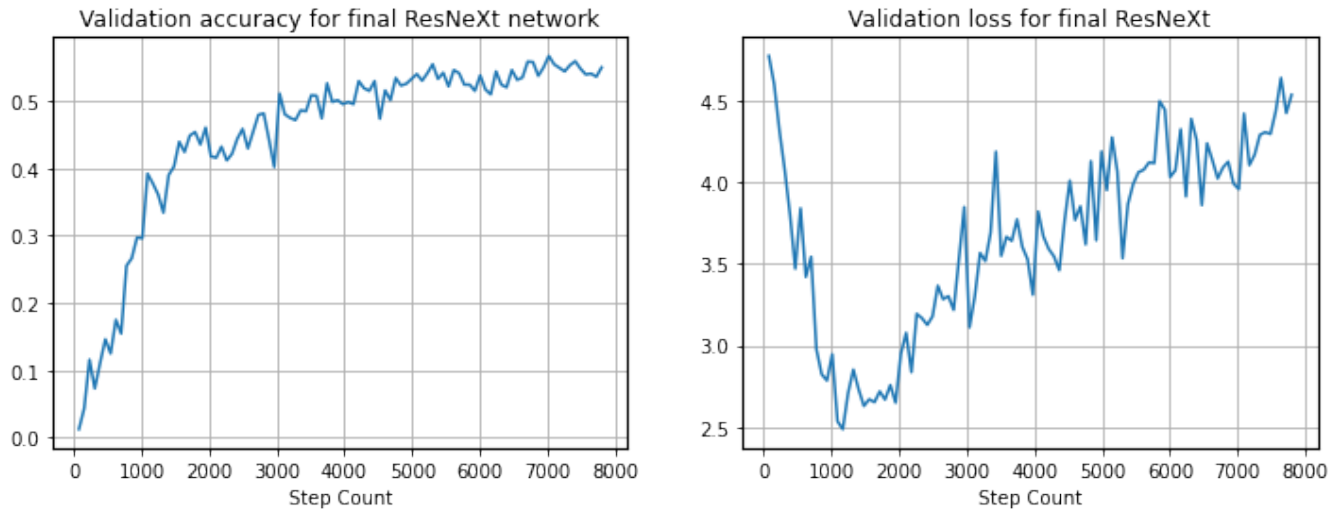

Figure 1: The validation accuracy and the negative log likelihood validation loss values for the ResNeXt architecture with non-augmented data.

4.1 Hyperparameter pilot study

For ResNet10 and ResNet18, the observation of overfitting can be attributed to the learning rate being too large. A smaller learning rate and larger batches result in more stable learning. Furthermore, these results show that, for optimization, learning rate decay might be useful to quickly reach a lower loss, whilst keeping stability in later epochs. When comparing the testing EER scores in Table 3 and Table 4, a conclusion can be made that deeper networks do not always perform better. One possible cause is that the dataset size is too small for the our implementation of the more complex ResNet18 network. A network with more trainable parameters benefits from more data.

4.2 Data augmentations

The training the networks making use of the augmented dataset was optimized based on the pilot study. Learning rate decay was introduced for more stable learning in later epochs based on the results in the pilot study, where the loss became progressively more unstable. Furthermore, the number of epochs was increased to ensure the model would converge to make comparable trainings with and without augmentations. The largest batch size in the pilot study was kept for computational efficiency and to prevent overfitting on the small dataset.

The efficiency of the network was greatly improved by compiling an offline library of augmented data that could be directly used during training. Furthermore, there was some improvement on training accuracy from the data augmentation in ResNet10 and ResNeXt, which positively alter the data with more varying quality. The robustness induced by augmentations such as noise and reverberation possibly are not as applicable to these high quality recordings as they would be in a real life setting where the environment is less controlled. Furthermore, the data augmentation was implemented as one block instead of judging the individual effect of each augmentation. It is possible that the use of only some augmentations would have yielded better results. Similarly, another area of improvement could have been the tuning of each augmentation's parameters Data augmentations should have been done separately to determine their individual effects on the performance of the models. For further improvement one should not only do a separate analysis of the different augmentations, but also optimize the individual parameters within the augmentations. Furthermore, one could have made sure there was no class imbalance, and subsequently configured this to improve accuracy.

4.3 Final submission

Our ResNeXt implementation yielded our best testing EER result of 19.03. But the loss curve in Figure 1 shows overfitting. This means that, given more time, the hyperparameters could have been optimized better. The learning rate shows instability in later epochs. Thus less epochs and learning rate decay would have been the next step.

Although the data augmentation showed promising results it was not included in the final submission. The decision was primarily attributed to time constraints and limitations in our planning, such as the offline data augmentations having to be run locally without the opportunity to transfer the library between group members due

to storage restrictions, which prevented us from assessing the effectiveness of the data augmentation on the final model. Secondly, the smaller online data augmentations we did try with the final submission network showed no improvement. This counter intuitive result could be because the data augmentations performed were irrelevant with the chosen parameters.

The inability to include results and effects of all data augmentations pose a real limitation to our project as the final model could have returned different accuracy, loss and EER with data augmentation.

5 Conclusion

The aim of this project was to build an optimally performing speaker detection system. Through a pilot study we analysed the effects of hyperparameters on our implemented architectures. We found that certain combinations of hyperparameters show overfitting. Using these findings we further experimented with ResNet10, ResNet18 and ResNeXt architectures with data augmentation to see the effects. This experimenting showed that data augmentation improves the accuracy of our models and the testing EER of ResNet10 and ResNeXt. Based on these results we tried to optimize ResNeXt and managed to outperform the baseline prototype model. The final testing EER achieved was 19.03.

References

[1] Brownlee, J. Train neural networks to reduce overfitting, Dec 2018.

[2] Cai, W., Chen, J., and Li, M. Exploring the encoding layer and loss function in end-to-end speaker and language recognition system, 2018.

[3] Chung, J. S., Huh, J., and Mun, S. Delving into voxceleb: environment invariant speaker recognition, 2019.

[4] Chung, J. S., Huh, J., Mun, S., Lee, M., Heo, H. S., Choe, S., Ham, C., Jung, S., Lee, B.-J., and Han, I. In defence of metric learning for speaker recognition, 2020.

[5] Chung, J. S., Nagrani, A., and Zisserman, A. Voxceleb2: Deep speaker recognition, 2018.

[6] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (June 2016), IEEE.

[7] Nagrani, A., Chung, J. S., Xie, W., and Zisserman, A. Voxceleb: Large-scale speaker verification in the wild. *Computer Speech & Language 60* (Mar. 2020), 101027.

[8] Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. Don't decay the learning rate, increase the batch size, 2017.

[9] Xie, S., Girshick, R., Dollar, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (July 2017), IEEE.

[10] Zhang, A., Lipton, Z. C., Li, M., and Smola, A. J. Dive into deep learning. *arXiv preprint arXiv:2106.11342* (2021).

[11] Zhong, Q., Dai, R., Zhang, H., Zhu, Y., and Zhou, G. Text-independent speaker recognition based on adaptive course learning loss and deep residual network. *EURASIP Journal on Advances in Signal Processing 2021*, 1 (July 2021).

6 Appendix

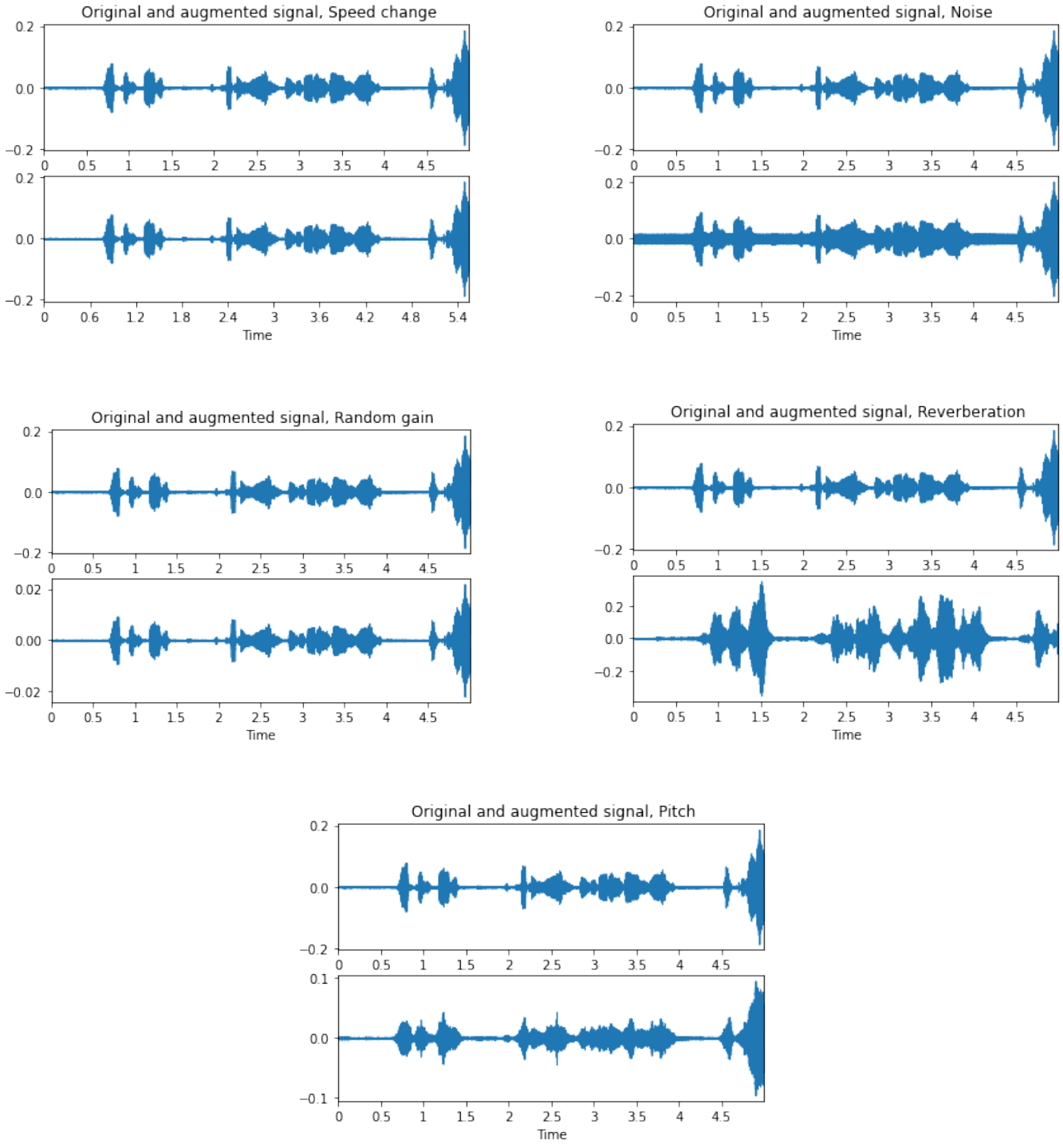

Figure 2: An example audio clip from the training dataset illustrating the change between the original data (top) and the augmented data (bottom) for each of the implemented augmentations. Note that the scale of the x axis (time) is different for the speed augmented signal, and the scale of the y axis (amplitude) is different for the random gain augmented signal.

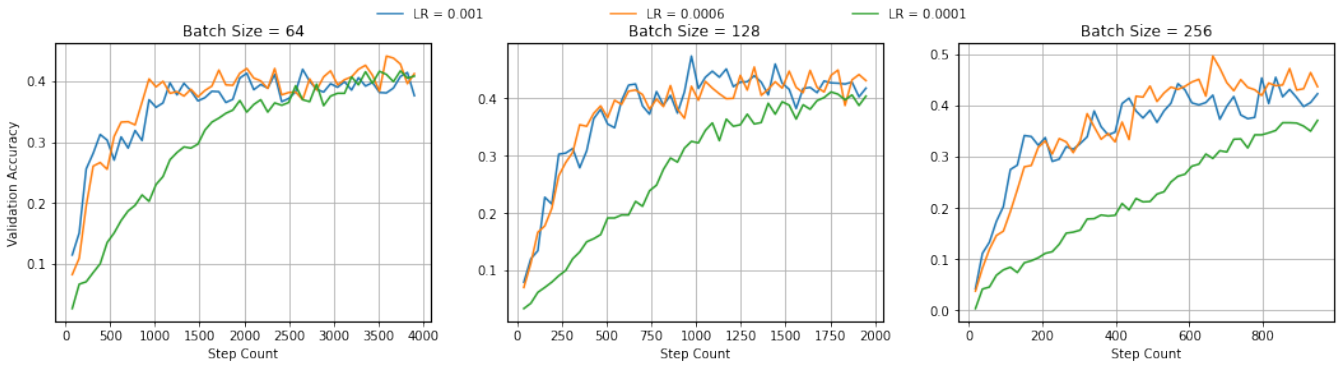

Figure 3: Validation accuracy values for the ResNet10 architecture. Each instance was trained for 50 epochs. Each line specifies values for a certain learning rate at the specified batch size.

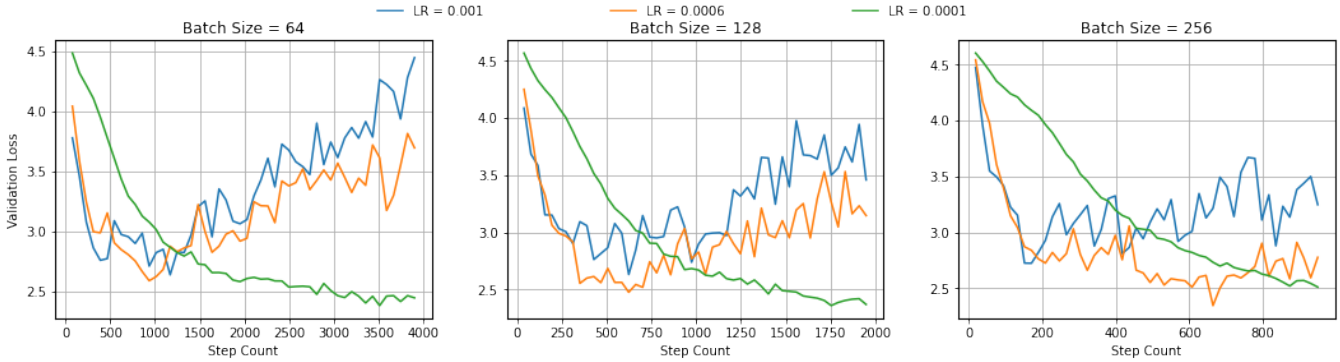

Figure 4: The negative log likelihood validation loss values for the ResNet10 architecture. Each instance was trained for 50 epochs. Each line specifies values for a certain learning rate at the specified batch size.

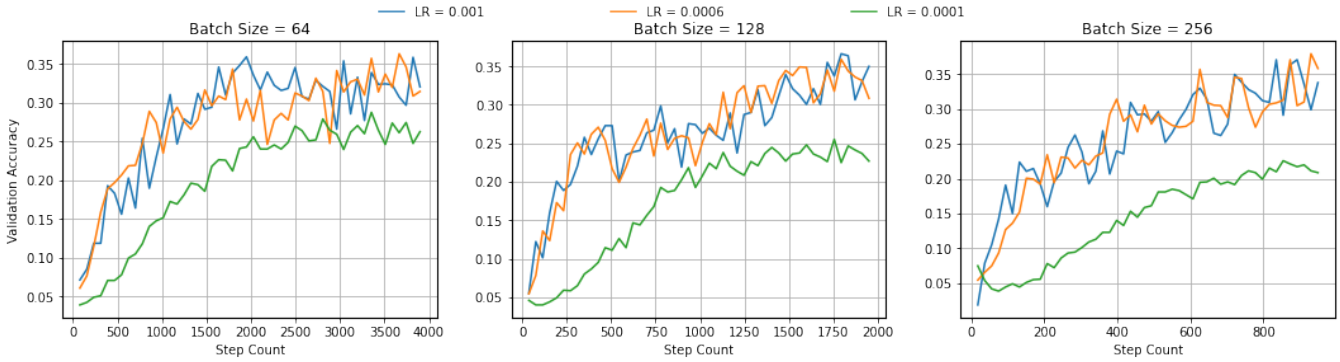

Figure 5: Validation accuracy values for the ResNet18 architecture. Each instance was trained for 50 epochs. Each line specifies values for a certain learning rate at the specified batch size.

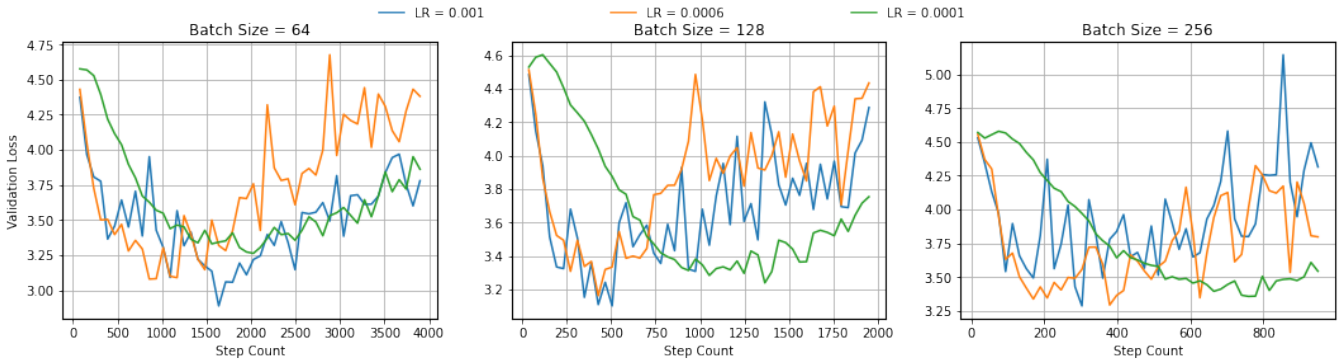

Figure 6: The negative log likelihood validation loss values for the ResNet18 architecture. Each instance was trained for 50 epochs. Each line specifies values for a certain learning rate at the specified batch size.

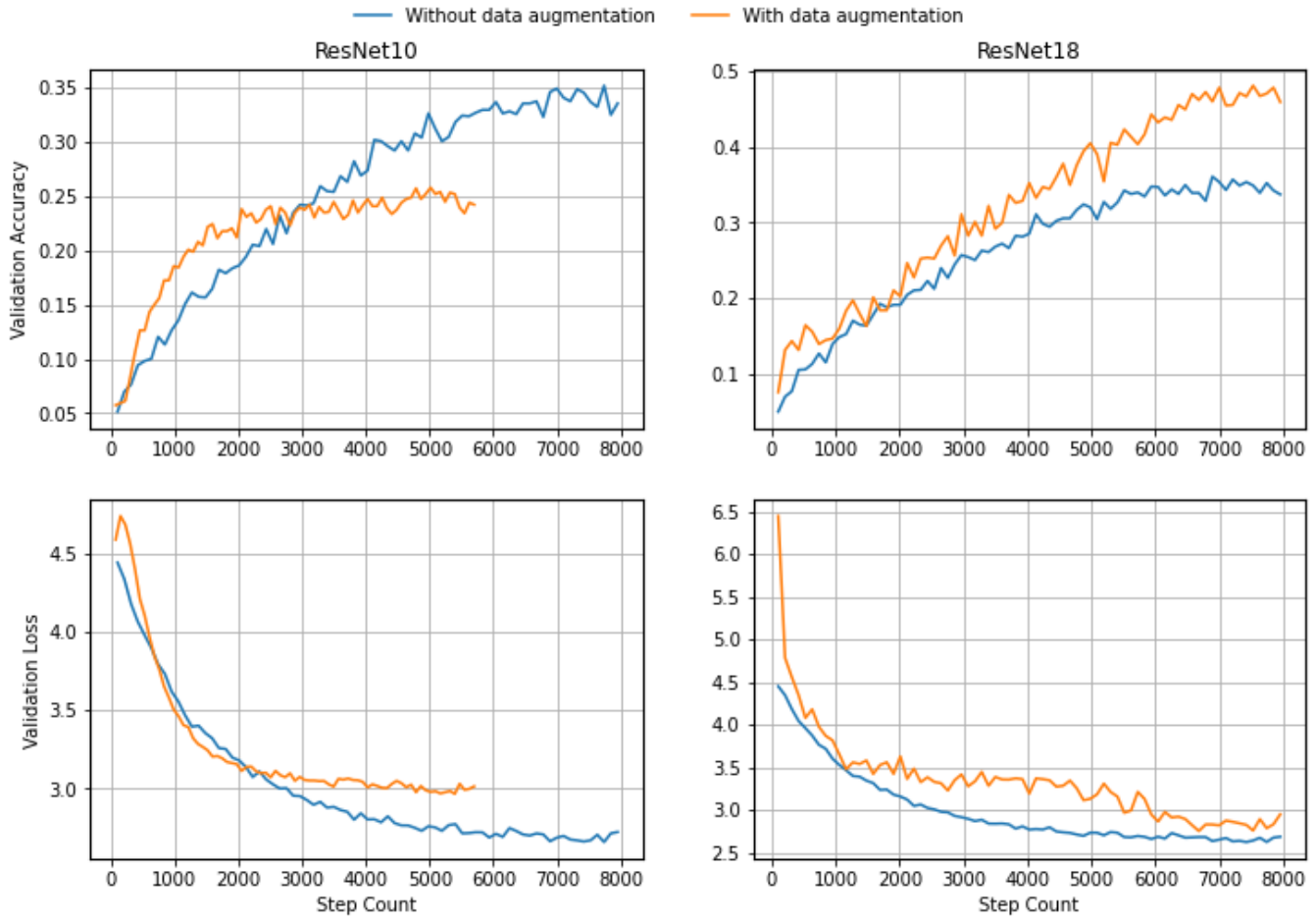

Figure 7: The validation accuracy and validation loss for ResNet10 and ResNet18 when run with and without data augmentations. Number of epochs = 75, learning rate = 0.00015, batch size = 256, $\gamma = 0.95$ and decay after every epoch.

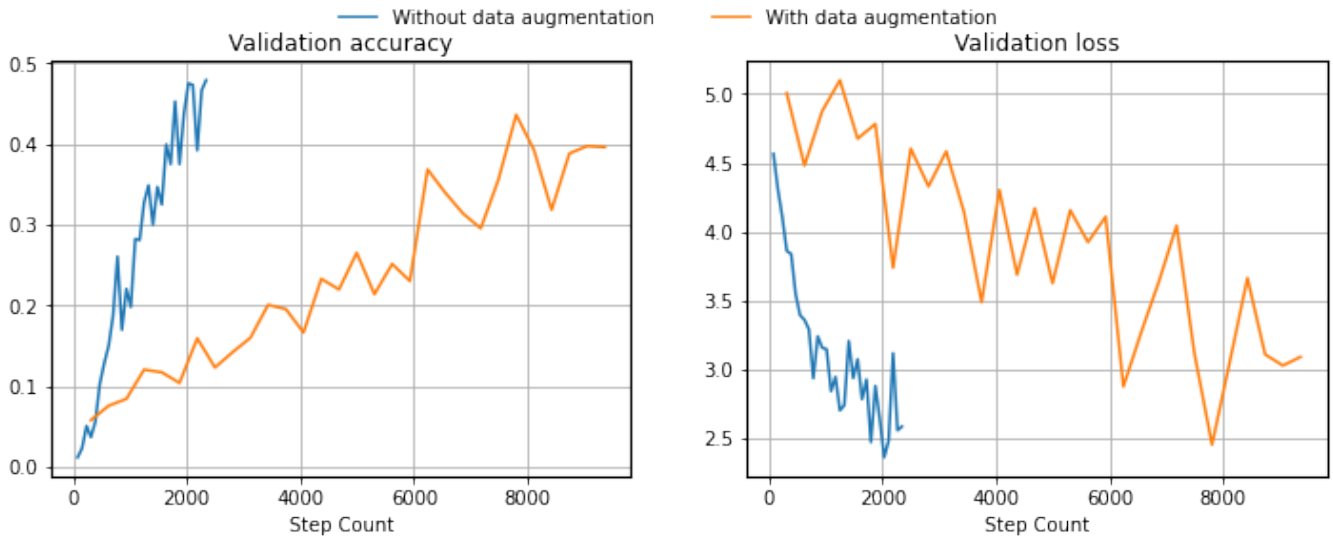

Figure 8: The validation accuracy and validation loss for ResNeXt when run with and without data augmentations. Number of epochs = 30, learning rate = 0.005, batch size = 64, $\gamma = 0.85$ and decay after every 10th epoch.

Table 3: Testing EER results for the ablation study on ResNet10.

Batch size	Learning rate	Testing EER
64	0.001	24.52 %
64	0.0006	24.73 %
64	0.0001	24.95 %
128	0.001	25.84 %
128	**0.0006**	**24.14 %**
128	0.0001	25.60 %
256	0.001	25.01 %
256	0.0006	25.04 %
256	0.0001	25.57 %

Table 4: Testing EER results for the ablation study on ResNet18.

Batch size	Learning rate	Testing EER
64	**0.001**	**27.00 %**
64	0.0006	27.55 %
64	0.0001	37.55 %
128	0.001	27.68 %
128	0.0006	31.26 %
128	0.0001	40.59 %
256	0.001	29.33 %
256	0.0006	28.25 %
256	0.0001	40.72 %