SMALL-FOOTPRINT ACOUSTIC SCENE CLASSIFICATION THROUGH 8-BIT QUANTIZATION-AWARE TRAINING AND PRUNING OF RESNET MODELS

Technical Report

Laurens Byttebier, Brecht Desplanques, Jenthe Thienpondt, Siyuan Song, Kris Demuynck, Nilesh Madhu

IDLab, Department of Electronics and Information Systems,
Ghent University - imec, Belgium
{brecht.desplanques, jenthe.thienpondt, siyuan.song, kris.demuynck, nilesh.madhu}@ugent.be

ABSTRACT

This report describes the IDLab submissions for Task 1a of the DCASE Challenge 2021. The challenge consists of constructing an acoustic scene classification model with a size of less than 128 KB. All submitted systems consist of a ResNet based model enhanced with Squeeze-and-Excitation (SE) blocks trained with temporal cropping, time domain mixup and speed-change augmentation strategies. Grouped convolutions are incorporated in all models to reduce the model complexity. Three submissions are based on 8-bit quantization-aware training with a fusion of batch norm and convolutional layers to reduce the parameter count even further. Further, two of these three systems explore multi-class score calibration by means of multinomial or one-vs-rest logistic regression. The calibration is then fused with the final linear output layer of the network to avoid an increase in model size. The fourth submission explores parameter pruning on a model with 16-bit weights as an alternative to the 8-bit weight quantization. The uncalibrated 8-bit model outperforms the pruned 16-bit model slightly and achieves a log loss of 0.82 and an accuracy of 71.2% on the standard test set of the TAU Urban Acoustic Scenes 2020 Mobile development dataset.

***Index Terms*—** acoustic scene classification; residual neural network; quantization; pruning; score calibration

1. INTRODUCTION

As many of the applications of Acoustic Scene Classification (ASC) are targeting mobile hardware nowadays, there is an increasingly urgent need for classification models with a limited computational complexity and memory footprint. Based on this insight, Task 1a of the DCASE 2021 Challenge [1] consists of designing an ASC model constrained to a storage size of 128 KB for its model parameters. This choice to compare the systems based on the minimal required storage space rather than on their throughput, latency or memory requirements enables a fair comparison of the different systems, as this is independent of the gains that may accrue from the specific hardware the models run on or the coding language used.

In this report, several parameter reduction techniques will be explored to decrease the number of parameters in a RESNET-based [2] ASC model. Apart from direct optimization by reducing the layer widths and thus the number of channels in each layer, we apply weight quantization [3], grouped convolutions [4, 5] and pruning [6]. We also explore prediction score post-processing that directly optimizes the log loss criterion through logistic regression based calibration [7, 8]. This extra calibration step does not have an impact on the number of model parameters, as the regression weights can be fused with the final linear layer of the ASC model. All RESNET models are implemented with *PyTorch Lightning* [9], and the logistic regression is trained with *scikit-learn* [10].

This report is structured in four sections. In section II the baseline system architecture will be introduced. Section III details the methods to transform the baseline systems into small footprint models. In section IV the configuration of the four submitted systems is discussed, together with the corresponding results on the development data.

2. BASELINE SYSTEM

2.1. Training setup

We use the suggested training and test partitions of the TAU Urban Acoustic Scenes 2020 Mobile development dataset [11]. There are 13965 train audio clips and 2970 test audio clips. The dataset defines 10 different acoustic scenes and 9 recording devices. The test partition is balanced on the level of both these recording properties.

During training we optimize the cross-entropy loss with the Adam optimizer [12]. We use a Cyclical Learning Rate (CLR) schedule and train the model for three full cycles with the *triangular2* policy [13]. The maximum and minimum learning rates are equal to 1e-3 and 1e-6, respectively. The maximum learning rate decays with a factor of 2 after each full cycle of 120 epochs. We also impose a weight decay value of 1e-4.

We apply on-the-fly data augmentation by taking 3 second random temporal crops of each training recording. Each crop is resampled with a rate of either 85%, 100% and 115% of the original sample frequency of 44.1 KHz with equal probability. After padding to restore the original length of the crops, we apply Mixup [14] in the time domain by additive mixing of the 64 crops within a mini-batch. We set $\alpha = 0.2$ in the enforced beta distribution of the mixing weights.

2.2. Feature extraction

We use the log amplitudes of the Mel-FilterBank Energies (MFBE) of short, overlapping signal segments as the input features. The window size is set to 30 ms and the frame shift is 10 ms. The time-domain data is first transformed into the Fourier domain after applying a Hann window and computing a 4096-point DFT. The DFT output magnitudes for the positive frequency spectrum are then mapped to the output of a 128-point Mel-filter bank. We apply mean nor-

malization to the log MFBE features of each crop across time before it is input to the neural network.

2.3. Model architecture

As Convolutional Neural Networks (CNNs) achieved promising ASC results in the DCASE setup [11] we will focus on the proven residual network (RESNET) [2] architecture. This is a type of CNN that utilizes extra skip connections between the layers to speed up the training by reducing the vanishing gradient problem. The proposed architecture is shown in Figure 1. To allow for audio input of variable duration we incorporate a pooling layer that estimates the temporal mean and standard deviation of the input feature maps to obtain a fixed-length representation. This representation is mapped by a final linear layer to 10 output nodes. The activations of the output nodes can be sent through a softmax layer to estimate the final acoustic scene probabilities of the input recording.

The pre-pooling frame-level layers consist of convolutional layers. All frame-level layers except for the input layer are a *Double Conv Block* which incorporates two convolutional layers with each a kernel size of 3×3. The block is displayed in Figure 2. The stride modifies the amount of movement when sliding the convolutional filter across the time and frequency axis. The frequency stride has a direct impact on the number of output statistics of the pooling layer. The stride can thus be used to limit the number of required connections in the final linear layer. The second layer in the Double Conv Block consists of grouped convolutions with a cardinality of 4 to limit the number of parameters. See Section 3.2 for more details. The convolutions operate locally with a limited receptive field in both time and frequency space to generate the output activations. To exploit global information as well in the frame-level layers we incorporate Squeeze-and-Excitation (SE) blocks [15]. These blocks rescale the activations per channel according to information contained in a global descriptor. See Figure 3 and [15] for more details. The number of nodes in the SE bottleneck is set to 4 for the first three Double Conv Blocks. The final three blocks use a bottleneck size of 8 in the SE module.

All Double Conv Blocks except for the first one are bridged by a skip connection. Due to the relatively low number of channels we choose to include a linear projection in all skip connections. The linear projections are also used to compensate for dimensions mismatches between the input and output of the Double Conv Block. These dimension mismatches are caused by strides larger than one and differences between the number of input channels and output channels in the underlying convolutional layers.

The input size of the final dense layer depends on the number of input Mel-filterbanks, the total frequency stride of the preceding layers, the number of output channels in the final frame-level layer and the fact that the pooling layer produces both a mean and standard deviation estimate per feature. To be more specific, for the architecture depicted in Figure 1 we obtain an input dimension of $(128 \,/\, 2^3) \times 48 \times 2$ for the final dense layer that produces 10 output activations.

2.4. Calibration

The primary evaluation metric of DCASE 2021 Task 1a is the log loss. Neural networks can be poorly calibrated [8], which will have a negative impact on the log loss on unseen data. In addition, we have a mismatch between crop duration during the training phase (3 s crops) and test phase (10 s crops). In order to calibrate the output

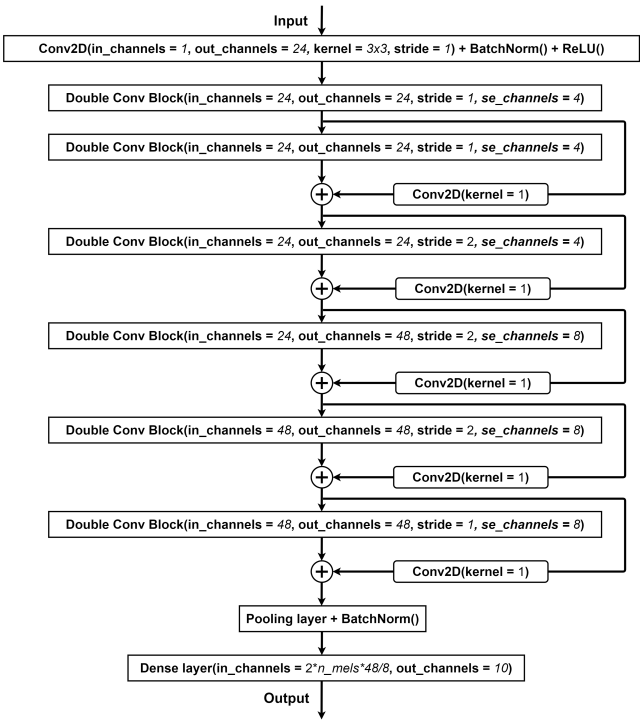

Figure 1: The RESNET architecture used in all submission. Note that the fourth submission with pruning reduces the channel size from 24 (48) to 20 (40) in the Double Conv Blocks.

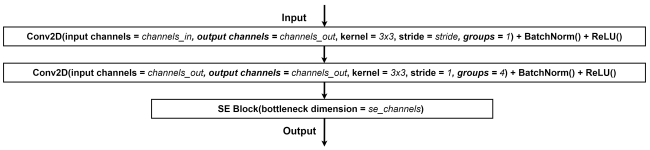

Figure 2: A *Double Conv Block* with SE-module. Note that that the first convolution layer can apply a stride > 1. Only the final convolutional layer uses grouped convolutions with a cardinality of 4.

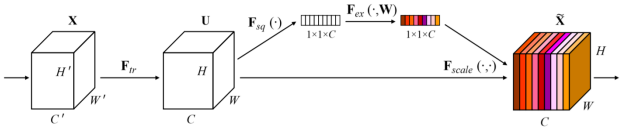

Figure 3: A Squeeze-and-Excitation block [15] that incorporates global information in the frame-level layers.

scores we can train a logistic regression model [7]. We train the logistic regression model on the acoustic scene logit output scores extracted from the non-augmented test partition of the DCASE2020 development dataset, utilizing full length clips. This calibration step should allow the model to produce more reliable soft decisions on unseen data. We do not fit an intercept, and allow for different scaling factors for all classes and scores. This results in total of 10×10 scaling factors that can be easily integrated in the final layer of the neural network. We explore both multinomial and one-vs-rest multiclass logistic regression to estimate the calibration parameters. Note that the calibrated models produce optimistic scores on the test partition of the development data, and these results may not permit a fair comparison with other systems.

3. SMALL FOOTPRINT STRATEGIES

Several methods can be applied to reduce the total model parameter size below 128 KB. A straightforward way of decreasing the number of parameters is to reduce the width and the depth of the neural network. A reduction of the width corresponds with the lowering of the number of channels in the convolutional layers. A reduction of the depth of the network corresponds with the deletion of convolutional layers.

A large proportion of the numbers of parameters in the proposed architecture occurs in the final dense layer. Reducing the number of weights in this layer will hence cause a notable reduction in the overall amount of parameters. Applying multiple frequency strides in the frame-level layers larger than one and/or lowering the amount of frequency features in the input will reduce the input dimension of the dense layer. Note that the temporal dimension has no impact on the model size due to the temporal statistics pooling layer.

Three other, more advanced, methods to reduce the size of the networks consist of weight quantization [3], grouped convolutions [4, 5] and pruning [6]. See below for more details.

3.1. Quantization

Although quantization does not reduce the total number of parameters, it reduces the memory required to store them by reducing the floating point precision. Most deep learning frameworks store floating point values in a 32-bit format. By applying quantization, the model parameters can be converted into 16-bit floating points or even 8-bit integers. A downside of this method is the inherent reduction in classification performance (although this degradation is not as pronounced for 16-bit floating point numbers [16, 17]) and the increased risk for numerical overflows.

Deploying 8-bit integer quantization can mainly be performed in the following three ways: *dynamic quantization*, *post-training static quantization* and *quantization-aware training* (QAT) [18, 3]. For 8-bit quantization we rely on QAT, as an 8-bit model trained with QAT should be able to reach very similar classification performance to that of an identical trained 32-bit model. During the training phase, QAT operates on a 32-bit model but it mimics the 8-bit quantization that will occur post-training. During inference the quantized 8-bit model is used. As is common with QAT, we fuse the convolutional operation with the Batch Norm (BN) and RELU activation to reduce the number of parameters slightly.

32-bit floating points within a certain range and centered around a midpoint can be cast to 8-bit integers by the following conversion:

$$x_q = round\left(\frac{x_j}{s_q} + z_q\right) \quad (1)$$

where x_q denotes the 8-bit quantized value corresponding with floating point value x_j, s_q is the corresponding scale and z_q is the midpoint (or zero point) of the range.

For the 8-bit quantization of the convolutional filter weights, different scales s_q are determined per channel for each filter and these scales are stored in 64-bit floating point numbers in PyTorch. We assume that the weights are symmetrically distributed around zero and enforce z_q for every channel to be zero. The additive operations in the convolutions are performed with 32-bit precision [3] and thus the biases of the convolutional layer are still stored in 32-bit precision. In this approach a re-quantization step is required after each quantized convolution layer. This requantization step requires the storage of a single scale in 32-bit and a single zero point in 64-bit. Similar quantization steps with a single scalar and zero point are required at the start of the network, the additive operation of all skip connections and at the end of each network module that operated in 32-bit precision (all SE-blocks and the statistics pooling layer).

The 8-bit quantization of the weights in the final dense layer again requires a 64-bit scale per output channel (10 scales). z_q is again zero for every channel. The biases are stored in 32-bit. This final layer also utilizes a re-quantization step on the output activations with a single 32-bit scale and a single 64-bit zero point.

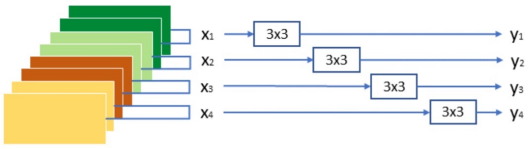

Figure 4: A grouped convolution with 4 groups (cardinality=4) [19].

3.2. Grouped convolutions

Grouped convolutions reduce the parameter count in the network by dividing the input and output channels into smaller groups. A separate smaller kernel is determined to map each input channel group to its corresponding output channel group. This reduces the number of required kernel weights by a factor equal to the number of groups. This number of groups is also called the cardinality of the grouped convolution. See Figure 4 and [19] for more details. These grouped convolutions are used in the final convolutional layer of the Double Conv Blocks.

3.3. Pruning

Pruning sets weights of parameters which are expected to have a small impact on the model performance due to over-parametrization to zero. Typically this is done for weights that have small magnitudes or L1-norms. The operation can be implemented in two ways: either weights with a magnitude below a certain threshold are pruned or pruning is applied to a *proportion* of all weights [20]. The latter will be used in this report.

The non-pruned randomly initialized network is first trained for 1 CLR cycle. In between the first and second CLR cycle, unstructured L1-norm based pruning is applied the network to set a certain

portion of the weights to zero. As this pruning decreases the accuracy, the network training is continued for 2 more CLR cycles to recover.

4. SUBMISSION MODELS AND RESULTS

This section describes the different strategies used in the four submitted systems to acquire a total model parameter size below the 128 KB limit. Additionally, some systems use a separate calibration stage based on logistic regression. The performance of all submitted systems on the test fold of the DCASE development set can be found in Table 1.

- **System 1 (8-bit quantized) - qat_8b**: To reduce the parameter size of the baseline model to the requirements of the DCASE Challenge, we apply 8-bit weight quantization combined with QAT. All weights are quantized except for the layers in the SE-blocks and the statics pooling layer, due to numerical instability encountered during inference.
- **System 2 (8-bit quantized + multi. calib.) - 8b_calibm**: This submission uses a separate calibration stage to map the output scores of the baseline system to more reliable probabilities. We use the test partition of the provided DCASE development set to train a multinomial logistic regression classifier on the output scores of system 1. The calibration weights are merged with the final quantized linear layer of the model, resulting in the same model size as the uncalibrated system.
- **System 3 (8-bit quantized + OVR calibration) - 8b_calibo**: Another calibration strategy is explored in this submission. Again, we use the test partition of the development set of the DCASE Challenge to create a calibration set based on the output scores of system 1. A separate binary logistic regression classifier is trained for each class in a one vs. rest (OVR) fashion. Subsequently, the output logits of all classifiers are converted to probabilities using the standard softmax function. Note that this approach deviates from the standard OVR approach to apply a sigmoid function on the output of each classifier and to normalize the output probabilties across the different classes. Finally, the calibration weights are merged with the last linear layer to prevent increasing the model size.
- **System 4 (16-bit quantized + pruned) - 16b_prune**: The fourth submission is a model trained using a pruning strategy in full 32-bit precision. The model has a reduced width of 20 output channels in the initial layer and the first three Double Conv Blocks and 40 channels in the final three Double Conv Blocks. The model is trained for one full cycle, after which we set 25% of the weights with the lowest magnitude to zero. Subsequently, the model is trained further until convergence without updating the pruned weights. After the training stage, we quantize the parameters to 16 bit. No calibration is applied after training.

4.1. Model complexity

System 1 - 3 use the same model and quantization strategy. Since the extra parameters of the calibration stage are fused with the final linear layer, this results in an equal parameter count as shown in Table 1. Because of the 8-bit quantization these models with a size 127.64 KB can fit 113976 non-zero parameters. The pruning and 16-bit quantization strategy of System 4, which uses the same baseline model architecture, fits 62390 non-zero parameters in 121.86 KB.

System	# non-zero parameters	Log loss	Accuracy (%)
System 1	114.0K	0.82	71.2
System 2*	114.0K	(0.66)	(75.3)
System 3*	114.0K	(0.71)	(74.1)
System 4	62.4K	0.84	70.2

Table 1: ASC performance of the submitted systems (<128 KB) on the standard test fold of the TAU UrbanAcousticScenes 2020 Mobile development dataset, along with the number of non-zero parameters. * denotes that these system results cannot be used for system comparisons due to calibration on the test set.

4.2. Results

As shown in Table 1, the 8-bit QAT strategy results in the best performance on the test fold of the DCASE 2021 development dataset with a log loss score of 0.82. However, the pruning strategy results in a much smaller parameter count with only a small performance drop with a log loss score of 0.84. The reported performance of System 2 and 3 cannot be used for system comparisons as we calibrated these systems on the test partition of the TAU UrbanAcoustic Scenes 2020 Mobile development dataset.

5. REFERENCES

[1] I. Martín-Morató, T. Heittola, A. Mesaros, and T. Virtanen, "Low-complexity acoustic scene classification for multi-device audio: analysis of dcase 2021 challenge systems," 2021.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[3] D. S. Khudia, J. Huang, P. Basu, S. Deng, H. Liu, J. Park, and M. Smelyanskiy, "FBGEMM: enabling high-performance low-precision deep learning inference," *CoRR*, vol. abs/2101.05615, 2021. [Online]. Available: https://arxiv.org/abs/2101.05615

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, 2012, p. 1097–1105.

[5] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 5987–5995.

[6] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=rJl-b3RcF7

[7] J. C. Platt, "Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods," in *Advances in Large Margin Classifiers*. MIT Press, 1999, pp. 61–74.

[8] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger, "On calibration of modern neural networks," in *International Conference on Machine Learning*. PMLR, 2017, pp. 1321–1330.

[9] W. Falcon *et al.*, "Pytorch lightning," *GitHub. Note: https://github.com/PyTorchLightning/pytorch-lightning*, vol. 3, 2019.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[11] T. Heittola, A. Mesaros, and T. Virtanen, "Acoustic scene classification in dcase 2020 challenge: generalization across devices and low complexity solutions," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2020 Workshop (DCASE2020)*, 2020, submitted. [Online]. Available: https://arxiv.org/abs/2005.14623

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[13] L. N. Smith, "Cyclical learning rates for training neural networks," 2017.

[14] H. Zhang, M. Cissé, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond empirical risk minimization," *CoRR*, vol. abs/1710.09412, 2017. [Online]. Available: http://arxiv.org/abs/1710.09412

[15] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," *CoRR*, vol. abs/1709.01507, 2017. [Online]. Available: http://arxiv.org/abs/1709.01507

[16] D. Das, N. Mellempudi, D. Mudigere, D. D. Kalamkar, S. Avancha, K. Banerjee, S. Sridharan, K. Vaidyanathan, B. Kaul, E. Georganas, A. Heinecke, P. Dubey, J. Corbal, N. Shustrov, R. Dubtsov, E. Fomenko, and V. O. Pirogov, "Mixed precision training of convolutional neural networks using integer operations," *CoRR*, vol. abs/1802.00930, 2018. [Online]. Available: http://arxiv.org/abs/1802.00930

[17] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *CoRR*, vol. abs/1502.02551, 2015. [Online]. Available: http://arxiv.org/abs/1502.02551

[18] R. Krishnamoorthi, J. Reed, M. Ni, C. Gottbrath, and S. Weidman, "Introduction to quantization on pytorch," March 2020, consulted on May 19th, 2021 [Online]. [Online]. Available: https://pytorch.org/blog/introduction-to-quantization-on-pytorch/

[19] T. Zhou, Y. Zhao, and J. Wu, "Resnext and res2net structures for speaker verification," 2020.

[20] D. W. Blalock, J. J. G. Ortiz, J. Frankle, and J. V. Guttag, "What is the state of neural network pruning?" *CoRR*, vol. abs/2003.03033, 2020. [Online]. Available: https://arxiv.org/abs/2003.03033