LOW-COMPLEXITY ACOUSTIC SCENE CLASSIFICATION USING ONE-BIT-PER-WEIGHT DEEP CONVOLUTIONAL NEURAL NETWORKS

Technical Report

Mark D. McDonnell

Computational Learning Systems Laboratory,
UniSA STEM,
University of South Australia, Mawson Lakes SA 5095, Australia

ABSTRACT

This technical report describes a submission to Task 1b ("Low-Complexity Acoustic Scene Classification") in the DCASE2020 Acoustic Scene Challenge. Solutions for this task were required to be constrained to have parameters totalling no more than 500 KB. The strategy described in this report was to train a deep convolutional neural network applied to spectrograms formed from the acoustic scene files, such that each convolutional weight was set to one of two values following training, and hence could be stored using a single bit. This strategy allowed a single 36-layer all-convolutional deep neural network to be trained, consisting of a total of 3,987,000 binary weights, totalling 486.69KB. The model achieved a macro-average accuracy (balanced accuracy score) across the three classes of 96.6±0.5% on the 2020 DCASE Task 1b validation set.

***Index Terms*—** acoustic scene classification; one-bit-per-weight; deep residual network; log-mel spectrograms

1. INTRODUCTION

Task 1 in the DCASE2020 IEEE AASP Challenge on Detection and Classification of Acoustic Scenes and Events required entrants to design classifiers that predicted the scene label of ten-second recordings. The setup was similar to that in 2018 [1] and 2019. Models were to be trained and validated using a development set, and tested and evaluated on an evaluation set.

Task 1 was divided into two subtasks. This technical report is relevant only to Task 1b: "Low-Complexity Acoustic Scene Classification." This task requires classification of audio scenes into three classes: indoor, outdoor and transportation. Models are constrained such that "Classifier complexity for this setup is limited to 500KB size for the non-zero parameters." The 500KB constraint excludes any deep neural network batch normalization layer parameters used, and an parameters used for feature extraction.

Python code using tensorflow.keras for training our models and running trained models is at `https://github.com/McDonnell-Lab/DCASE2020_Task1b`.

We did not use any data other than the 40 hours comprised from 14400 ten-second (24 bit, 48 KHz) stereo recordings provided by the challenge organizers.

2. STRATEGY

Highest accuracy in previous approaches to scene classification have arisen from treating spectrograms of acoustic scenes as though they are images, and training deep Convolutional Neural Networks (CNNs) on these spectrograms using best practice image classification methods, e.g. [2, 3, 4]. We also adopt the use of a CNN applied to spectrograms.

Past DCASE Challenges (and other machine learning contests) tend to be won by ensembles combined using either simple averaging, or by meta-learning approaches involving stacking. We did not concentrate our efforts on this aspect, instead preferring to seek the best (and largest within the 500 KB constraint) single network we could.

2.1. One-bit-per-weight CNNs

Past CNNs for acoustic scene classification, such as that reported in [4], have used in the order of 1–10 million convolutional weights, which by default are represented using 32 bit floating point precision. For example. a model with 4 million weights and 32 bits per weight would total 15,625 KB. Meeting the 500 KB constraint would allow for only in the order of 125,000 parameters, which is a very small deep CNN.

To enable a typical size of deep CNN, we therefore limit the number of bits per weight, when stored on disk, to just a single bit. Doing this permitted us to design a CNN that otherwise closely resembled a good unconstrained design for this dataset, i.e. one with 36 convolutional layers and a total of 3,987,000 convolutional weights.

The technique we used to enable the storage of weights with a single bit was precisely that described in our previous work [5, 6]. The following paragraph and Figure 1 are quoted from [5]:

"...the only differences we make in comparison with full-precision training are as follows. Let $\mathbf{W}_i$ be the tensor for the convolutional weights in the i–th convolutional layer. These weights are processed in the following way only for forward propagation and backward propagation, not for weight updates:

$$\hat{\mathbf{W}}_i = \sqrt{\frac{2}{F_i^2 C_{i-1}}}\,\mathrm{sign}\left(\mathbf{W}_i\right), \quad i = 1, \ldots, L, \tag{1}$$

where F_i is the spatial size of the convolutional kernel in layer i..." and C_i is the number of input channels to a layer.

It needs to be emphasised that this change applies only for propagation through layers (forwards and backwards during training,

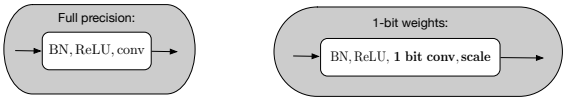

Figure 1: **Difference between our full-precision and 1-bit-per-weight networks.** The "1 bit conv" and "scale" layers are equivalent to the operations shown in Eqn. (1). [Figure from from [5]]

and forwards for inference). During training, weight updates are made using float32 weight tensors. These are discarded for inference, and during inference, a sign operation need not be applied if weights are already binarized (which is how there are intended to be for stored on disk). The scaling factors can be considered as a part of the model itself rather than as parameters, since their values depend only on the number of input channels to a layer, and the size of a layer's kernel.

Our keras implementation for training using one-bit-per-weight convolutions can be observed in our repository at `https://github.com/McDonnell-Lab/1-bit-per-weight` – see `class BinaryConv2D` in `ResNetModel.py`. This is repeated in our challenge repository at `https://github.com/McDonnell-Lab/DCASE2020_Task1b`.

2.2. Class balancing minibatch sampling strategy

We noticed that the development data did not contain equal class counts; indoor and transportation had 4320 samples while outdoor had 5760 (for the offical training split, this reduced to 2704 indoor, 3754 outdoor and 2724 transportation). Although this is not badly imbalanced, we decided to implement a form of class balancing as follows.

Training was subdivided into epochs, such that in each epoch the total number of samples used was equal to $3 \times \min_i(N_i)$, where N_i is the number of samples in the i–th class. Each sample from the minority class was used once in each epoch, and the remaining samples were chosen as a random subset from each class at the start of each epoch. Given our use of mixup augmentation, where each sample fed to the neural network is comprised from two training set samples, the number of samples fed to the neural network in one epoch was $1.5 \times \min_i(N_i)$.

As we used relatively small minibatches (32 samples), we also tried to ensure a minibatch was as balanced as possible. At the start of each epoch we randomly shuffled within each class the indices of the samples chosen for that class for the epoch. We then created minibatches in a sequence of 3 samples at a time, such that one sample was taken in a random order from each class.

We found in validation that this class balancing approach increased our accuracy by about 2 percent. We found the use of class weightings in the loss function or at inference time were unable to achieve the same accuracy as the use of this class balancing method.

2.3. Acoustic feature extraction

Past entries into DCASE challenges have used a range of approaches to forming image-like spectrograms for CNN processing. These have included log-mel spectrograms, MFCCs, perceptual weighted power spectrograms, CQT spectrograms and so forth.

We used only log-mel spectrograms, calculated for both left and right channels, so that the input to our CNN had 2 channels.

2.4. CNN design

As in our past work [4], we note that spectrograms have characteristics different from optical images [7]. For example, one object placed behind another is entirely occluded in a photograph, whereas sounds from two sources superimpose such that frequency features in a spectrogram can arise from a combination of the two sources. Another important difference is that an object can appear anywhere in an image, and carry the same meaning, whereas patterns of features at low frequencies may represent different physical origins from those at higher frequencies. Consequently the time and frequency axes that comprise the two axes of a spectrogram are not of the same nature as the two spatial axes in an image. Thirdly, frequency features at any point in time can be non-local, due, for example, to harmonics.

The second and third point leads us to use a CNN in which the frequency and time axes are treated differently. The main deviations from a standard deep CNNs are described in Section 3.

While we have not observed generalizable improvements using this split, we have observed much faster training, and accuracy at least as good as networks without a split.

2.5. Aggressive regularization and data augmentation

We found significant levels of overfitting, i.e. the training loss and error rate for our trained models applied to the training set were close to zero for sufficiently large models. Therefore, we used several forms of regularization and data augmentation.

Like many entries in previous DCASE challenges, we used mixup augmentation [8], and like most deep CNNs for image classification, we used weight decay for all convolutional layers. We also experimented with shift-and-crop augmentation, but found best results when only relatively mild temporal cropping was used. Finally, we made use of a new approach from image classification which is not in common practice, which was to add a form of regularization where batch normalization layers did not have their offset and scale parameters learned [5].

Coupled with using these approaches, we found it helpful to train for a very large number of epochs (we used 310) in a warm-restart learning-rate schedule [9].

3. METHODS

All networks were trained using tensorflow.keras (version 2.2.4-tf), which is an integrated component of tensorflow (we used version 1.13.0).

3.1. Acoustic file preprocessing

To calculate our log-mel energies, we used 4096 FFT points, the original sampling rate of the acoustic files (48 KHz), frequencies from 0 to half of the sampling rate, a hop-length of 1024 (a quarter of the number of FFT points), and the HTK formula to define the mel scale [10]. Our implementation used python, and the LibROSA library[1]. Our resulting spectrograms were of size 469 time samples, and 256 frequency bins. Thus, the total size of the input tensor used for training our final model was $[14400, 256, 469, 2]$.

[1] `https://librosa.github.io/librosa/`

To avoid the need to recalculate spectrograms for every experiment, we created them once and saved to and load from disk.

3.2. Splitting of high and low frequencies

The CNN we designed is a residual network pre-activation variety [11]. It has two mostly parallel paths that combine only using late fusion by addition of frequency axes (unlike the concatenation of [4]), two convolutional layers before the network output. The overall network input has 256 frequency dimensions, but these are immediately split in the network such that frequency dimensions 0 to 125 are processed by a residual network with 17 convolutional layers and frequency dimensions 128 to 256 by another. All kernels in these paths are 3×3. After these stacks, the two pathways are added and then operated on by a 3×3 convolutional layer and then a 1×1 convolutional layer. The second of these layers reduces the number of channels to the number of classes, i.e. three. This is followed by a batch normalization layer, a global average pooling layer, and softmax.

See [4] for further discussion of this design.

3.3. No downsampling in frequency layers

The input to our network for training has 400 time samples (due to random temporal cropping—see below) and 256 frequencies. Due to the all-convolutional nature of the network, at inference time we can use larger number of time samples, and use all 469 samples provided by our audio preprocessing.

In order to ensure the CNN can learn global temporal information across all time samples, we use standard image classification practise of regularly downsampling in time using stride 2 convolutional layers. The principle is that an important cue could happen with equal likelihood at any point in time in a 10 second sample, just like objects in images can appear in any spatial location.

However, in the frequency axis we do not downsample. Consequently, the number of frequency dimensions in the feature maps for each path remains constant at 128 throughout the network. Hence, at the point where the two branches are added, each path has processed a frequency-axis receptive field of 35 dimensions.

Consequently, the global average pooling layer does not merge equal global "views" in the frequency axis, but instead averages over different overlapping views spanning 35 dimensions, rather than global views.

3.4. Other CNN design aspects

Our CNN is of the residual network pre-activation variety [11], which means it has the input to each convolutional layer first processed by a batch normalization layer and then a ReLU activation. In the residual paths, when the number of channels needs to be increased before summation of different paths, we used zero padding in the channel dimension as in [5], rather than 1×1 convolutions.

Using a technique introduced in [5], the very first layer of our network was a batch normalization layer with learned offset and scale parameters. This enabled us to avoid assumed forms of normalization of the features passed into the network.

3.5. Regularization and data augmentation

We used the following:

- **weight decay**: we used an aggressively large value of 2.5×10^{-4} (i.e. 5×10^{-4} when set in keras) on all convolutional layers.
- **Not learning batch normalization scale and offset**: it was shown in [5] that for datasets and networks with significant overfitting, disabling the learning of batch normalization scale and offset (except in the very first layer) has a regularization effect resulting in improved test error rates on the CIFAR-100 benchmark. We used this approach here for all batch-normalization layers except at the input and the final batch normalization layer prior to the output. We decided to enable learning at the output, as part of our class balancing strategy.
- **Mixup and temporal crop augmentation:** As found by others in past DCASE challenges, we found it very useful to use mixup augmentation, using the same approach as [2], with $\alpha = 0.4$. We additionally used crop augmentation in the temporal axis: each of the two samples combined using mixup were first cropped independently and randomly from 469 dimensions down to 400.
- **Channel swapping:** We found about 0.5% improvement on the official DCASE2020 development set validation split if, during training, we randomly switched the left and right channels for each use of a sample, and then at inference averaged the raw predictions from each of the two possible channel orientations.

3.6. Training

We trained using backpropagation and stochastic gradient descent, with a minibatch size of 32, momentum of 0.9, and the categorical cross-entropy loss function. Each network was trained for 310 epochs using a warm restart learning rate schedule that resets the learning rate to its maximum value of 0.025 after $10, 30, 70$ and 150 epochs, and then decays according to a cosine pattern to 2.5×10^{-6}. It was shown by [9] and verified by [5] that this approach can provide improvements in accuracy on image classification relative to using stepped schedules.

3.7. Inference

At inference time, we used a low-complexity method of running the trained CNN only twice (once for each possible channel orientation), each time on the entire 469 sample spectrograms, and averaged the results, before finding the class with maximum response.

3.8. Validation

We noted that the evaluation dataset for this challenge includes two cities for which no data is provided during training. In order to try and design a model that generalised best to new cities, we used "leave-one-city-out' (LOCO) cross-validation during development of our system. As there were 10 cities in the training set, this meant for each model design that we needed to train and validation 10 different models. Each such model was trained on 9 of the cities, and validated on the 'left-out' city. We made decisions on many aspects of the design based on whether average accuracy across the 9 LOCO models increased or decreased.

An official train/validation split of the DCASE development data was provided for Task 1b, roughly in a 70:30 ratio. We finalised the design and selected model selection using this split, and

then retrained our model using the entirety of the development data before running the models on the evaluation data for submission.

4. RESULTS

Our best results on the official contest validation splits of the Task 1b development set are shown in Table 1, with a confusion matrix shown in Figure 2. Note that for individual classes in this table, 'accuracy' means 'recall' (equivalently, 'sensitivity'). But for overall, accuracy is the total number correct divided by the total number of samples, rather than average recall.

Table 1: Results for Task 1b on official development validation split of 4185 samples.

Data	Accuracy	Log Loss
indoor	94.7%	0.149
outdoor	97.8%	0.082
transportation	98.9%	0.054
Overall	97.1%	0.094

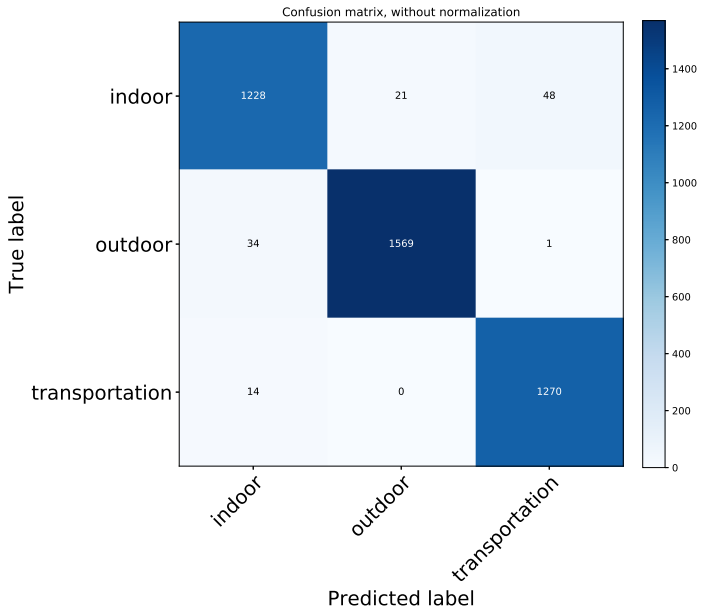

Figure 2: **Task 1b validation set confusion matrix.**

The DCASE200 Task 1b challenge is evaluated using accuracy calculated as the average of the class-wise accuracy, also known as 'balanced accuracy' or 'macro-average accuracy.' Given the development set validation split has unequal numbers within each class, this means balanced accuracy is not exactly equal to the raw classification accuracy. However, we generally found both metrics to usually give very similar numbers.

5. DISCUSSION

For each subtask, four submissions were permitted. We submitted results for four independently trained versions of precisely the same system. We made this choice because during validation we observed larger-than-typical variability in validation performance from independently runs. We hope that one of our four runs is at the upper end of an accuracy spread in a window of about $\pm$ 0.5%, and expect that at least one run will be above the mean.

We note that the class pair with least confusions during validation was 'outdoor' and 'transportation.' 'Indoor' had comparable numbers of confusions with 'outdoor' and 'transportation.'

In LOCO validation, Barcelona consistently achieved lowest accuracy, while Paris tended to be highest.

6. REFERENCES

[1] A. Mesaros, T. Heittola, and T. Virtanen, "A multi-device dataset for urban acoustic scene classification," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE2018)*, November 2018, pp. 9–13. [Online]. Available: https://arxiv.org/abs/1807.09840

[2] Y. Sakashita and M. Aono, "Acoustic scene classification by ensemble of spectrograms based on adaptive temporal divisions," Tech. Rep., 2018, DCASE 2018 technical reports.

[3] M. Dorfer and G. Widmer, "Training general-purpose audio tagging networks with noisy labels and iterative self-verification," in *Proceedings of the Detection and Classification of Acoustic Scenes and Events 2018 Workshop (DCASE2018)*, November 2018, pp. 178–182.

[4] M. D. McDonnell and W. Gao, "Acoustic scene classification using deep residual networks with late fusion of separated high and low frequency paths," in *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2020, pp. 141–145.

[5] M. D. McDonnell, "Training wide residual networks for deployment using a single bit for each weight," 2018, in Proc. ICLR 2018; arxiv: 1802.08530.

[6] M. D. McDonnell, H. Mostafa, R. Wang, and A. Schaik, "Single-bit-per-weight deep convolutional neural networks without batch-normalization layers for embedded systems," in *2019 4th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS)*, 2019, pp. 197–204.

[7] D.Rothman, "What's wrong with CNNs and spectrograms for audio processing?" Tech. Rep., 2018, https://towardsdatascience.com/whats-wrong-with-spectrograms-and-cnns-for-audio-processing-311377d7ccd.

[8] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "Mixup: Beyond empirical risk minimization," in *International Conference on Learning Representations*, 2018.

[9] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with restarts," *CoRR*, vol. abs/1608.03983, 2016. [Online]. Available: http://arxiv.org/abs/1608.03983

[10] S. J. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, *The HTK Book Version 3.4*. Cambridge University Press, 2006.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," Microsoft Research, Tech. Rep., 2016, arxiv.1603.05027.