Recreating Environment Sound Classification Using a Two-Stream CNN Based on Decision-Level Fusion

1st Robert Sparks
rs16175@bristol.ac.uk

2nd Alfred Brown
ab16230@bristol.ac.uk

I. Introduction

The purpose of this paper is to replicate four convolutional networks proposed by Su et al. in their 2019 paper on environmental sound classification (ESC) [1]. Their paper presents a model termed TSCNN-DS that applies a late fusion method using Depster-Shafer evidence theory. It consists of two parallel networks, the LMCNet and the MCNet, which share the same architecture, however differ in their input features. The proposed TSCNN-DS model reaches accuracies of 97.2% on the UrbanSound8k dataset. For comparative purposes a fourth network (MLMC) is presented that uses as its input the union of the features of the LMCNet and MCNet. Four, six and eight convolutional layer versions of the different networks are explored in their paper.

For this paper, the same UrbanSound8k dataset and auditory features were used for training and testing of the networks. However, instead of using ten-fold cross-validation, one-fold validation was applied and instead of applying Depster-Shafer evidence theory for late fusion, a softmax and averaging method was used. We therefore refer to the late fusion network in this paper as TSCNN. Furthermore, only the four layer convolutional networks were considered in the attempted replications.

II. Related Work

Various different environmental sound classification (ESC) systems have been built over the years, but it was only until Piczak [2] used a convolutional neural network (CNN) for this task in 2015 that CNNs became a research field for ESC classification. Since then, different CNN structures and input features have been explored (see [3] for example), and the classification accuracy has climbed steadily upwards. Current accuracies even surpass those of the paper we are attempting to replicate, reaching accuracies of 98.77% for the UrbanSound8k dataset [4].

III. Dataset

The UrbanSound8K dataset consists of recordings of sounds found in daily urban environments. Specifically, the UrbanSound8k dataset contains 8732 labelled sound excerpts from ten different classes: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, and street music. The sound excerpts are less than 4 seconds long and were accumulated from the website https://freesound.org/. This is a free resource and sounds are uploaded by users across the globe.

Since any user can upload anything, the recordings vary in their quality. This could potentially lead to loss of some key information in some of the features. Furthermore, each class has varying quantities of so-called foreground and background data (salience), implying some data will be noisier than others, which may also affect the class accuracies [5].

IV. Input

Five different auditory features are extracted from this dataset and used in different combinations to train and validate different networks. These auditory features extract sound information and transform them into discrete $H \times W \times 1$ tensors. For simplicity, the features are abbreviated as follows, and their dimensions (height vs width) are also shown:

$$\left.\begin{matrix}\text{Chroma } (7 \times 41) \\ \text{Spectral } (12 \times 41) \\ \text{Tonnetz } (6 \times 41)\end{matrix}\right\} \text{CST } (25 \times 41)$$

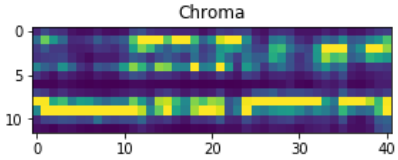
$$\left.\text{Log-Mel Spectrogram}\right\} \text{LM } (60 \times 41)$$

$$\left.\text{Mel-Frequency Cepstral Coefficient}\right\} \text{MFCC } (60 \times 41)$$

LMCNet combines features LM and CST, MCNet combines features MFCC and CST, and MLMCNet combines all features, as their input. In contrast, TSCNN uses the LMCNet and MCNet in parallel and then fuses their output together.

Each feature is extracted from a segmented clip of an audio file. The features convert different aspects of the time-signal into a matrix of information. As an example, the Chroma feature is presented below.

Fig. 1: The matrix formed by extracting chroma data from a sound time-signal. With the other features these can be combined to form images.



V. Architecture

The convolutional networks were implemented with the following architecture. For clarity, the kernel sizes K, input I and output O dimensions, stride length S, and padding sizes P are presented for each convolution. For example, the first convolution operation has the following properties: $(K, I, O, S, P) = ((3,3), 1, 32, (1,1), (1,1))$, which implies a 3 by 3 kernel was used on a 1 channel input image 32 times with a stride of 1 x 1 and a padding size of 1 x 1. Similarly, for the max pooling layers, the kernel, stride and padding sizes are given in the form (K, S, P). The letter M is used to denote the size of the flattened output after layer 4. For the LMCNet and MCNet, $M = 15488$ and for the MLMCNet, $M = 26048$. For the 2D Dropout operations the dropout probabilities are presented as well. No bias was used.

- **LMCNet, MCNet, MLMCNet**: The structures of these networks are identical, the only difference lies in their input. LMCNet and MCNet have input dimensions of $85 \times 41 \times 1$, whereas MLMCNet has dimensions of $145 \times 41 \times 1$.

$$
\begin{aligned}
\text{L1} &\begin{cases} \text{2D Conv: } ((3,3), 1, 32, (1,1), (1,1)) \\ \text{2D Batch Normalisation} \\ \text{RELU} \end{cases} \\
\text{L2} &\begin{cases} \text{2D Conv: } ((3,3), 32, 32, (1,1), (1,1)) \\ \text{2D Batch Normalisation} \\ \text{RELU} \\ \text{2D Drop Out: } (0.5) \\ \text{2D Max Pooling: } ((2,2), (2,2), (1,1)) \end{cases} \\
\text{L3} &\begin{cases} \text{2D Conv: } ((3,3), 32, 64, (1,1), (1,1)) \\ \text{2D Batch Normalisation} \\ \text{RELU} \end{cases} \\
\text{L4} &\begin{cases} \text{2D Conv: } ((3,3), 64, 64, (1,1), (1,1)) \\ \text{2D Batch Normalisation} \\ \text{RELU} \\ \text{2D Drop Out: } (0.5) \\ \text{2D Max Pooling: } ((2,2), (2,2), (1,1)) \\ \text{Flatten 2D into 1D} \end{cases} \\
\text{L5} &\begin{cases} \text{1D Linear: } (M, 1024) \\ \text{1D Batch Normalisation} \\ \text{Sigmoid Activation Function} \\ \text{1D Drop Out: } (0.5) \end{cases} \\
\text{L6} &\{ \text{1D Linear: } (1024, 10)
\end{aligned}
$$

- **TSCNN**: The TSCNN network applies the architecture presented above in parallel for the LMCNet and MCNet and then fuses their predictions (logits). The fusion was implemented by normalising the appropriate logit files using *Softmax* and then averaging class-wise.

For comparative purposes the number of parameters and dimensions are presented in Table I. While Su et al. do not show all the parameters for each operation, and round to one decimal point, the number of parameters seem to be the same. The only discrepancy lies in the 3rd convolution operation and final linear operation, whereby our rounded parameter numbers (to one decimal place) would be 36.9k instead of 36.8k, and 10.3k instead of 10.2k, respectively.

TABLE I: A comparison of parameter values to Su et al.'s architecture and output dimensions. Note, B is the batch size, and the other variables have their previous meaning

Layer	**Number of Parameters**		**Output Shape**
	This paper	**Su et al.**	**[B,O,H,W]**
Conv2D	288	288	[32,32,85,41]
BatchNorm	64	-	[32,32,85,41]
ReLU	0	-	[32,32,85,41]
Conv2D	9,216	9.2k	[32,32,85,41]
BatchNorm	64	-	[32,32,85,41]
ReLU	0	-	[32,32,85,41]
Dropout2D	0	-	[32,32,85,41]
MaxPool2D	0	-	[32, 32, 43, 21]
Conv2D	18,432	18.4k	[32, 64, 43, 21]
BatchNorm	128	-	[32, 64, 43, 21]
ReLU	0	-	[32, 64, 43, 21]
Conv2D	36,864	36.8k	[32, 64, 43, 21]
BatchNorm	128	-	[32, 64, 43, 21]
ReLU	0	-	[32, 64, 43, 21]
Dropout2D	0	-	[32, 64, 43, 21]
MaxPool2D	0	-	[32, 64, 22, 21]
Flatten	0	-	[32,15488]
Linear	15,860,736	15.9k	[32, 1024]
Sigmoid	0	-	[32,1024]
Dropout	0	-	[32,1024]
Linear	10,250	10.2k	[32,10]
Total	15,936,170	15.9 M	

When comparing the dimensions to the figures and written architecture for the LMCNet, MCNet and MLMCNet, an inconsistency appears in Su et al.'s paper. In section 3.2 Su et al. explain that the stride length is set to 2 x 2 for the first two layers and max pooling with a 2 x 2 kernel is applied at the end of the second layer. However, unless the max pooling kernel stride step is set to 1 x 1, or the stride lengths of the first and second layers are set to 1 x 1, the dimensions do not correspond to those shown in their Figure 4. Given this ambiguity, it was decided to simply set the stride lengths for the first two convolutions to 1 x 1 and set the max pooling layer to have a 2 x 2 kernel with a stride of 2 x 2. This avoided dealing with different padding values.

VI. Implementation Details

The code developed for this task includes TensorBoard logs, training and validation loops, command line arguments, and average and per class accuracies. The following steps were undertaken in the attempted replication of the networks:

- *Customised dataloader*: The starter code was adjusted to load the data. To validate that the dataloader was working as expected, the data was visualised and compared to figures in [1].
- *Model*: The paper was read carefully to find inconsistencies and the architecture was replicated to the best of our understanding. The model was then implemented in Pytorch's

nn.sequential module, as this allowed for a clear overview of the different operations within layers.

- *Per file validation*: The model is trained on sound file segments and the validation step was performed on a per file basis. The logits for each segment were normalised with *softmax* and then summed to find the label prediction for any given file. This is how the accuracy was computed.
- *Training*: Saving and loading model parameters were implemented on a command line basis for ease of use, and allowed training to be continued after previous training. Hyperparameters and the optimiser were selected by carefully reading Su et al.'s paper. There is an inconsistency in the paper, which cites the optimiser as Adam, yet gives mention to momentum, which is not explicitly a parameter. It was assumed that when the authors specify momentum, they are referring to the β_1 parameter of Adam. The weight decay parameter for Adam is also not given, and so we used a typical value of $1e-5$. The model results displayed in this paper were trained for 50 epochs.
- *Fusion*: The TSCNN was implemented by loading in the pre-trained LMCNet and MCNet models, and then retrieving their per file logits. For each file, the appropriate logits were normalised with *Softmax* then summed with *argmax*, giving the TSCNN prediction for each file.
- *Visualising results*: The training and validation logs were saved for each model and visualised in TensorBoard. This allowed for qualitative analysis of the models.

VII. Replicating Quantitative Results

The following table contains the class-wise and average accuracies for each of the different replicated networks. Generally we noticed a gradual improvement in the accuracies, starting with LMCNet and ending with TSCNN (left to right), which attained the highest average accuracy. However, in the results presented here MLMCNet actually does the worst on average. The two classes that were best predicted in this training run, and generally, were the gun shot and the jackhammer. These two classes both have a high proportion of foreground data available [5], but we surmise that it is sharpness (narrow, but high amplitude) of the waveform generated by both a gun shot and jackhammer that allowed for their high predictability.

TABLE II: The class-wise and average accuracies for each of the neural networks.

Class	LMCNet	MCNet	MLMCNet	TSCNN
ac	0.36	0.52	0.52	0.49
ch	0.94	0.85	0.91	0.94
cp	0.96	0.87	0.91	0.92
db	0.66	0.77	0.81	0.70
dr	0.80	0.83	0.81	0.83
ei	0.72	0.70	0.48	0.71
gs	0.97	0.97	0.94	0.97
jh	0.99	0.98	0.99	0.99
si	0.65	0.63	0.60	0.64
sm	0.91	0.91	0.88	0.93
Avg.	77.30 %	78.85 %	76.70 %	79.21%

VIII. Training curves

Fig. 2: Presented are the accuracy and loss curves, both for validation and training for each of the networks apart from the TSCNN (which was only computed after the LMCNet and MCNet had been trained). The darker colours indicate training and the lighter colours indicate validation. Please note that the horizontal axis shows the *SummaryWriter* steps and not the epochs, each tick corresponds to 6.25 epochs.

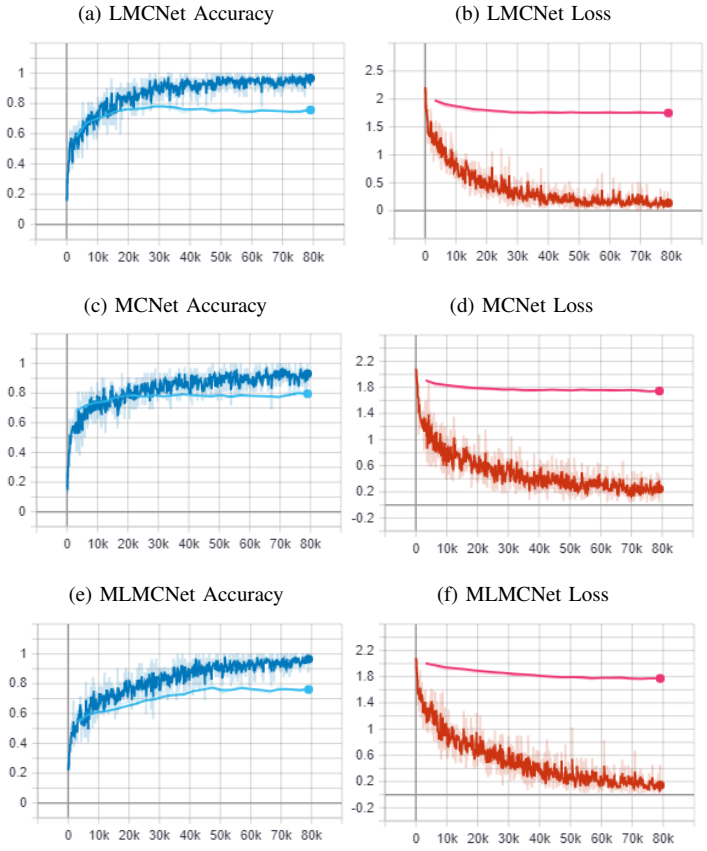


Figure 2 above shows the TensorBoard plots generated using the various networks. The training and validation accuracy of LMCNet increases until around 19 epochs (31k steps), where henceforth, the validation accuracy starts to decrease while the training accuracy continues to increase, until both start to converge asymptotically. The validation loss similarly starts to have zero gradient at around 19 epochs. This is indicative that the LMCNet is overfitting. The MLMCNet seems to suffer from the same issue, and starts to lose in validation accuracy at around 27 epochs. MCNet seems at first glance not to be overfitting, yet after many plateauing epochs, the validation accuracy seems to increase while the training accuracy decreases. This too is perhaps indicative of overfitting.

IX. Qualitative Results

This section presents various failure and success cases for each of the networks. For all cases, the MLMCNet input

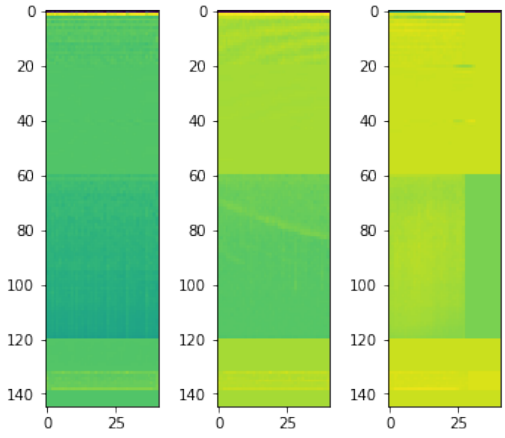
is presented as this contains both the LMCNet and MCNet input. In the presented images, the horizontal integers 0 to 59 show the MFCC feature that MCNet used, 6 to 119 show the LM feature that LMCNet used and 120 to 144 show the CST features that all networks used. Four different cases were explored: 1) correct prediction by LMCNet and MCNet, 2) correct prediction by LMCNet, false prediction by MCNet, and vice versa, 3) an example where the TSCNN outperforms LMCNet and MCNet, 4) an example where all networks failed.

1) Classes 'ac' and 'si' were chosen when searching for the example, because LMCNet and MCNet performed worst on these two classes. The examples are presented in Figure 3 (g) and (h), and it is particularly noticeable in plot (g) that both the MFCC and LM features have distinct lines going across them. In comparison, plot (n) (where all networks failed to predict the class appropriately) the matrix looks blurry, without any distinguishable lines. One might conclude from this that sound segments with strong features (edges, clustered discolourations, for example) within the input features are more likely to be predicted correctly by the neural networks.
2) Classes 'gs' and 'gh' were chosen for these cases, because both LMCNet and MCNet predicted them with the highest accuracy. Plots (i) and (j) show cases where LMCNet predicted correctly, and MCNet failed. Cases (k) and (l) show where the opposite was true. For cases (i) and (j), the defining LM feature shows large discolouration, while the MFCC appears manly uniform, perhaps explaining why LMCNet was able to predict this case and MCNet was not. In the cases where MCNet predicted correctly, the LM feature is blurry for the first case and striped in the second case, whereas MFCC has subtle horizontal lines for both. The LM features do not have the same discolouration at the same location as they did when LMCnet predicted correctly, which likely explains its false classifications for these two cases.
3) Class 'ei' was chosen, because TSCNN's class accuracy lay between that of LMCNet and MCNet. As plot (m) shows, the features MFCC and LM are both very blurry looking, perhaps indicating why LMCNet and MCNet were unable to predict this case.
4) Figure 3 (n) shows where all networks failed. The class 'jh' was chosen because it had the highest class accuracy. All the feature matrices appear blurry (the CST feature too), indicating a noisy sample, and perhaps, again, this explains why the networks failed to predict this particular case.

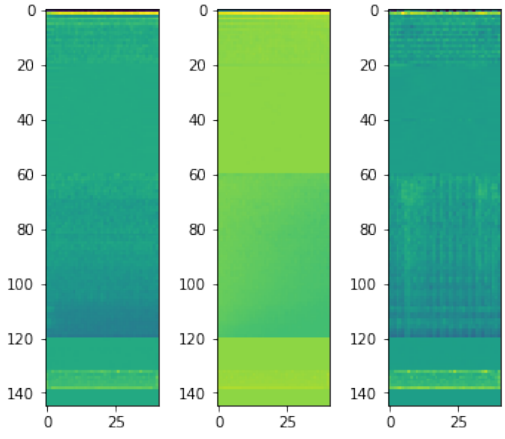
X. Improvements

As a potential improvement, Logmelspec augmentation was implemented as Salamon and Bello achieved state of the art results on the UrbanSound8K dataset. Pre-existing code was available on Github[1] and the most highly rated code was used and adapted for this task. Functions from the `librosa`

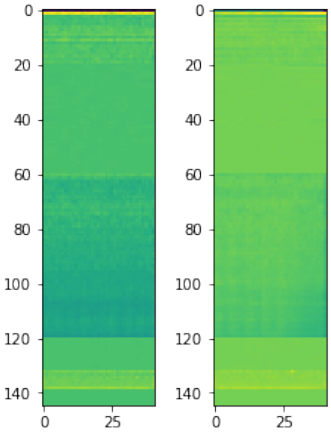
[1]Click here to see the repository [accessed 07.01.20] https://github.com/DemisEom/SpecAugment



(g) Class ac (h) Class si (i) Class gs



(j) Class jh (k) Class gs (l) Class jh



(m) Class ei (n) jh

Fig. 3: Presented are various success and failure cases: For (g,h) LMCNet and MCNet predicted these segments correctly; (i,j) LMCNet predicted these segments correctly but MCNet failed; (k,l) LMCNet failed to predict these segments correctly but MCNet succeeded; (m) shows one case where the TSCNN method outperformed LMCNet and MCNet; (n) shows an example case where all networks failed in their prediction. The .wav files in which these segments can, respectively, be found in: 119067-0-0-0.wav, 159742-8-0-0.wav, 200460-6-5-0.wav, 162134-7-12-2.wav, 169261-6-0-0.wav, 162134-7-12-2.wav, 83502-0-0-5.wav, 15544-5-0-8.wav, 34050-7-1-0.wav.

library were adapted as packages because they cannot be easily installed on Blue Crystal. The hyper-parameters were chosen by visualising the augmentation and comparing it to figures found in similar work that used augmentation in speech recognition [7]. The augmentation was performed in an online manner, giving a 50% probability to the augmentation of a sample. The results obtained are shown below.

TABLE III: The class-wise and average accuracies for each of the augmented neural networks.

Class	LMCNet	MCNet	MLMCNet	TSCNN
ac	0.42	0.62	0.48	0.56
ch	0.91	0.79	0.91	0.88
cp	0.88	0.88	0.97	0.85
db	0.80	0.76	0.69	0.78
dr	0.67	0.69	0.74	0.57
ei	0.51	0.57	0.60	0.51
gs	0.97	0.94	0.94	0.97
jh	0.96	0.97	0.92	1.00
si	0.53	0.61	0.60	0.67
sm	0.84	0.84	0.84	0.81
Avg.	72.28 %	75.51 %	73.24 %	73.60%

The augmentation decreased the average predictive performance for all of the networks, but it did increase some of the class accuracies. The results are likely due to a bad choice of hyper-parameters for the augmentation, which shows that our idea of trying to mimic the hyper-parameters used by [7] for speech-recognition was not the best approach. Augmentation has been shown to work by others for ESC classification [6], so it is not the augmentation that is at fault but rather our approach to the hyper-parameters.

XI. Conclusion and Future Work

This paper attempted to replicate four deep neural networks designed by Su et al.. The architectures of the networks were presented and inconsistencies in [1] were discussed, showing in detail the number of parameters and output dimensions of each operation within each layer of the networks. Our particular approach to overcoming these inconsistencies and our implementation of the networks were explained and the results presented in tabular form in Table II. The replicated neural networks produced average accuracies of 77.3%, 78.85%, 76.70% and 79.21%, for LMCNet, MCNet, MLMCNet and TSCNN, respectively. If early stopping would have been implemented, the networks would have achieved higher accuracies; for, as the qualitative results showed, all networks seemed to be overfitting. Moreover, particular failure and success cases were investigated, showing that the networks seemed generally only to fail on data that appeared noisy. Finally, an augmentation on the segments was explored, showing slight class accuracy improvements for some classes, but a general decrease in the average accuracies of the networks, which was likely due to ineffective hyper-parameter choices.

References

[1] Yu Su, Ke Zhang, Jingyu Wang, and Kurosh Madani. Environment sound classification using a two-stream CNN based on decision-level fusion. *Sensors (Switzerland)*, 19 (7):1–15, 2019. ISSN 14248220. doi: 10.3390/s19071733.

[2] Karol J Piczak. ENVIRONMENTAL SOUND CLASSIFICATION WITH CONVOLUTIONAL NEURAL NETWORKS. 2015. doi: 10.5281/zenodo.12714. URL https://github.com/karoldvl/paper-2015-esc-convnet.

[3] Naoya Takahashi, Michael Gygli, Beat Pfister, and Van Gool. Deep Convolutional Neural Networks and Data Augmentation for Acoustic Event Detection. Technical report. URL https://data.vision.ee.ethz.ch/cvl/ae.

[4] Jivitesh Sharma, Ole-Christoffer Granmo, and Morten Goodwin. Environment Sound Classification using Multiple Feature Channels and Attention based Deep Convolutional Neural Network. Technical report.

[5] Justin Salamon, Christopher Jacoby, and Juan Pablo Bello. A Dataset and Taxonomy for Urban Sound Research. doi: 10.1145/2647868.2655045. URL http://dx.doi.org/10.1145/2647868.2655045.

[6] Justin Salamon and Juan Pablo Bello. Deep Convolutional Neural Networks and Data Augmentation for Environmental Sound Classification. Technical report. URL https://github.com/justinsalamon/UrbanSound8K-JAMS.

[7] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le Google Brain. SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. Technical report.