Replication and Review of a Two-Stream Environment Sound Classification CNN Architecture

Angus Williams
aw16797@bristol.ac.uk

Rachel Kirby
rk16586@bristol.ac.uk

I. Introduction

Environment Sound Classification (ESC) is one of the core challenges involved in developing intelligent sound recognition (ISR) for identifying real world sound events. Many key innovations for advancing ISR technology have focused on the challenges of automatic speech recognition and music information recognition. Environmental audio events tend to be more diverse and disordered than speech or musical audio, therefore the application of these past innovations are often insufficient within ESC. Developing new models and frameworks designed specifically for the distinctive attributes of ESC tasks is a necessity for advancing ISR capabilities.

The research paper we have replicated [1] works towards this goal, focusing on the advancement of sound classification techniques for specific use within ESC tasks. Some key focuses of the paper include the extraction and manipulation of features, and the implementation of a CNN architecture.

The use of features such as Mel Frequency Cepstral Coefficients (MFCC) and Log-Mel Spectogram (LM) are common for sound classification tasks within automatic speech recognition, but are unsatisfactory for application within ESC. Su et al.'s paper, however, follows findings from related works where enhanced performance has been achieved through combining of these features.

Additionally, commonly applied methods such as Support-Vector Machines and Gaussian Mixture models do not offer sufficient time and frequency in variance for tackling ESC tasks. By developing a specialised a CNN approach, Su et al. work to overcome this issue.

II. Related Work

The use of CNN-based models for application within ESC tasks was first explored Karol J. Piczak, Ref. [2]. Using a model of 2 convolutional layers with max-pooling and 2 fully connected layers, Piczak was able to achieve a classification accuracy 5.6% higher than traditional approaches, demonstrating a potential for the use of CNN's within ESC tasks and calling for further research on the subject.

Other works have focused on improving the performance of CNN's in ESC tasks. In Ref. [3], Takahashi et al. propose a novel data augmentation technique to help reduce over-fitting of the model. Their technique involves mixing two acoustic sounds of the same class to create a new sound belonging to the same class. They found that their method provided more data variation and lead to a more robust model. This technique is not adopted by Su et al., but does provide insight as to possible further development of the model, to achieve a higher accuracy.

Perhaps the most influential work to the development of Su et al's research, however, is that of Chachada and Kuo, Ref. [4]. Their paper details the advantages of using aggregated features, as opposed to singular features, for improving performance within ESC tasks. Chachada and Kuo find that the combination of features is vital for capturing the non-stationary characteristics of environmental audio files.

III. Dataset

The dataset used by Su et al., and used within our replicated study, is the UrbanSound8K dataset [5] . The dataset contains 8,732 sound files, each less than 4s long, labelled in correlation to 10 sound classes: air conditioner, car horn, children playing, dog bark, drilling, engine idling, gun shot, jackhammer, siren, and street music. The audio files are in WAV format, which is an uncompressed raw audio format storing audio data, sampling rate, and bit rate.

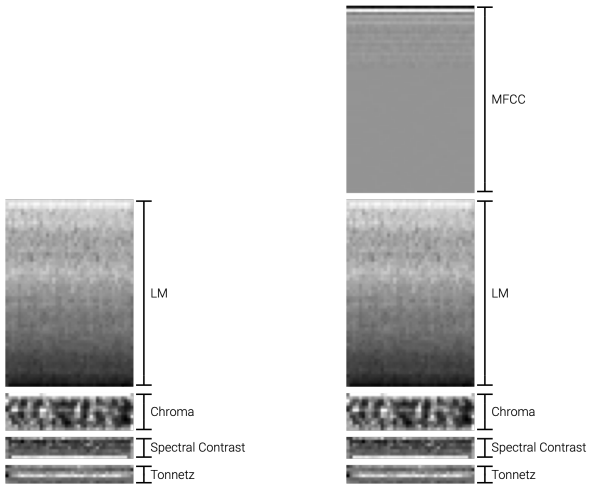
The audio files in the dataset are split into segments, with each segment corresponding to a file, and is classified the same as the file it originates from. For this experiment, the data set uses an approximate 9:1 test:train split, using 50,569 segments for training, and 5,395 segments for testing.

IV. Input

Each sample in the dataset is made up of multiple features derived from Mel filters, which we then combine to form the inputs to our neural network. The two main features are Log-Mel Spectrogram (LM), and Mel Frequency Cepstral Coefficients (MFCC), both of which were originally developed for automatic speech recognition, and so need to be combined with other features to successfully classify environment sounds.

We combine LM, Chroma, Spectral Contrast and Tonnetz to form the LMC input features; MFCC, Chroma, Spectral contrast and Tonnetz are combined to form the MC input features. We also combine MFCC, log-mel spectrum, chroma, spectral contrast and tonnetz to form the MLMC input features.

All features are represented as 2D matrices (LM:60x41, MFCC:60x41, Chroma:7x41, Spectral Contrast:6x41, Tonnetz:12x41) and are combined linearly (MC & LMC: 85x41, MLMC: 145x41). The resulting features can be represented as 2D Images. Some examples are visible in Fig.1.



(a) LMC (b) MLMC

Fig. 1: Visualisations of combined features

V. Architecture

We replicated the 4-conv neural network proposed by Su et al., with some changes, visible in Fig.2. The input and output sizes in Fig.2 are described for the LMC and MC inputs, and differ for the MLMC input. As specified in the paper, we used Cross Entropy Loss as our loss function, and Stochastic Gradient Descent as our optimisation function, and L2 regularization is achieved through batch normalisation. Dropout is set to 0.5 as per the original paper.

This architecture differs from that described by Su et al., due to inconsistencies in the authors' original paper.

Su et al. describe each of the four convolutional layers as utilising 2x2 stride, but also showcase in the original paper the size of the layers in Fig.2 and Fig.4, the number of parameters in the model in Table 1, as well as stating how "features with size of 11×22 are derived from the last hidden layer and feed to the fully-connected layer" in section 4.

2x2 stride causes the output size of a layer to be half that of it's input, and so using stride in every convolutional layer would cause cause huge dimensionality reduction, and would not match the sizes or number of parameters stated by the authors.

To resolve this, there are two possible solutions. In either case, the first 3 convolutional layers do not implement stride.

#	Layer Type	Num. of Kernels	Input Size	Output Size	Receptive Field	Batch Norm?	Dropout?	Activation Function
1	Convolutional	32	85x41x1*	85x41x32	3x3	Yes	No	ReLu
2	Convolutional	32	85x41x32	85x41x32	3x3	Yes	Yes	ReLu
3	Max Pooling	-	85x41x32	43x21x32	2x2 (Stride=2x2)	-	-	-
4	Convolutional	64	43x21x32	43x21x64	3x3	Yes	No	ReLu
5	Convolutional	64	43x21x64	43x21x64	3x3	Yes	Yes	ReLu
6	Max Pooling	-	43x21x64	22x11x64	2x2 (Stride=2x2)	-	-	-
7	Fully Connected	-	15488**	1024	-	Yes	Yes	Sigmoid
8	Fully Connected	-	1024	10	-	No	No	-

** 145x41x1 for MLMC (input/output sizes of subsequent layers also change for MLMC)*
*** 26048 for MLMC*

Fig. 2: Conv-4 Architecture Implemented

The fourth convolutional layer can either implement 2x2 stride to reduce the output size from 43x21 to 22x11, or not implement stride at all and instead apply 2x2 max pooling to reduce the size. Both approaches deliver the same layer sizes described and provide the same number of parameters by the authors. Having tested both approaches, we chose the max pooling solution, as it delivered higher accuracy.

The authors also only mention one fully connected layer in their model, despite the need for two fully connected layers. The first layer is needed to reduce the flattened result of the fourth convolutional layer to a 1024 width layer, and the second then reduce this to the final 10 width layer of results. As such, our architecture implements an additional fully connected layer.

VI. Implementation Details

A. Implement Working CNN Architecture

The first step of our implementation was to develop the network architecture, as mentioned in the previous section, testing a variety of methods for overcoming the various inconsistencies within the original paper. These adjustments were made within the CNN class initialisation and forward pass functions.

B. Implement Validation Process for Calculating Model Accuracy

After achieving a working version of the CNN architecture, we began to develop our validation process to obtain the model accuracy results.

We implemented two separate methods for calculating the accuracy of the models. One is focused on the accuracy calculation of the individual LMC, MC, and MLMC modes on the network. The second is written to calculate the accuracy after combining the results from LMC and MC within the TSCNN model.

For LMC, MC, and MLMC, we experimented with applying softmax to the logits obtained after running the validation set on the trained model. This re-calculated the scores of each segment, resulting in individual class predictions with a summation of 1. We used these scores to determine the class with the highest prediction, and then calculated the accuracy. We found, however, that our model obtained much higher accuracy results when we did not apply the soft max function, and instead calculated the predictions directly from the logits.

However, we did include softmax when determining the accuracy of the TSCNN model. We applied softmax on the logits determined from the LMCNet and MCNet, and then calculated the average of these LMC and MC scores to combine the two features.

We then implemented the Per Class Accuracy (PCA) calculation, which involved averaging the predictions scored for each segment in each file, and then calculating the accuracy of the resulting class prediction for each file. We tested our models multiple times to observe the amount of variation of results, and produce an average result for use within this report.

C. Testing efficiency of various parameters

We varied a few parameters to determine which were most efficient at producing the highest accuracy scores. These parameters included the number of epochs we trained our model on, we determined that 50 epochs was the most efficient as there was no noticeable accuracy improvement after this.

We also experimented with the Weight Decay used for optimisation in Stochastic Gradient Descent, testing the results after setting the decay at 0.01, 0.005, 0.001, 0.0005, 0.0001. 0.00005, 0.00001. We found 0.0001 (1e-4) reliably delivered higher accuracy results, and so we decided to settle on this value.

Finally, we tweaked our implementation of dropout within our CNN architecture. Whilst we initially used 1D dropout, we found the model achieved higher accuracy using 2D dropout. 2D dropout zeroes out randomly chosen channels input from a convolutional layer, this tends to be more effective at promoting independence of feature maps than 1D dropout [6].

VII. Replicating Quantitative Results

The results we produced were produced after 50 epochs, with our optimisation weight decay set to 0.0001. The results are visible in Fig.3. Our results largely fall within the 5% range, albeit for LMC, for which our algorithm proved to be less effective.

The late fusion TSCNN model appears to be the most effective on this dataset overall, although only performs better than it's peers on 2 of the individual classes, suggesting that is more generalised than the other models.

Class	LMC	MC	MLMC	TSCNN
ac	61.0%	78.0%	55.0%	78.0%
ch	93.9%	78.79%	90.9%	90.9%
cp	85.0%	89.0%	83.0%	85.0%
db	70.0%	72.0%	78.0%	72.0%
dr	87.9%	79.9%	82.8%	84.9%
ei	57.0%	78.5%	82.8%	78.5%
gs	90.6%	87.5%	96.9%	96.9%
jh	99.0%	92.7%	99.0%	99.0%
si	56.6%	54.2%	59.0%	60.2%
sm	85.0%	78.0%	84.0%	86.0%
Avg	***78.6%***	***78.9%***	***81.1%***	***83.1%***

Fig. 3: Replicated Results

VIII. Training curves

The training curves for LMC, MC and MLMC can be seen in Fig.4-Fig.9.

In Fig.5, our validation loss is clearly a lot higher than our training loss (over three times) and actually increases over time. Significant over-fitting has clearly occurred in the model for LMC.

In Fig.7, our validation loss is also higher than our training loss, so again we can assert that over-fitting has occurred in the model for MC. This is discounting the potential anomaly that occurs at the the end, wherein the loss suddenly jumps to around 1.1, and accuracy (in Fig.6) suddenly drops.

In Fig.9, again our validation loss is significantly larger than our training loss, so again we can assert that over-fitting has occurred for MLMC.

As we can observe some form of over-fitting in each of these cases, and the validation accuracy curves are relatively flat, we can infer that over training has occurred. This could have been remedied through a reduction in learning rate or in the number of epochs used, or through extension and augmentation of the data set.

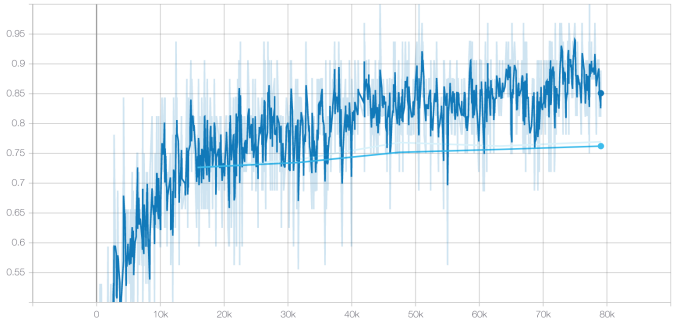


Fig. 4: LMC Accuracy

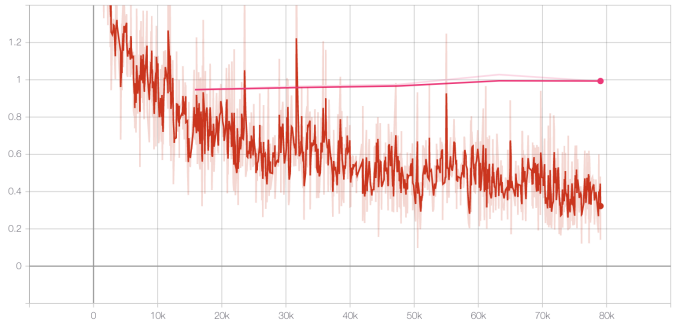


Fig. 5: LMC Loss

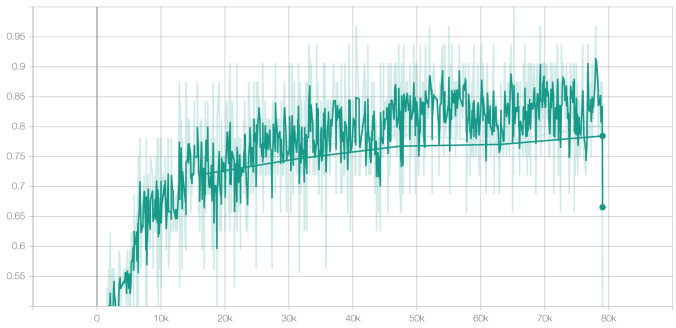


Fig. 6: MC Accuracy

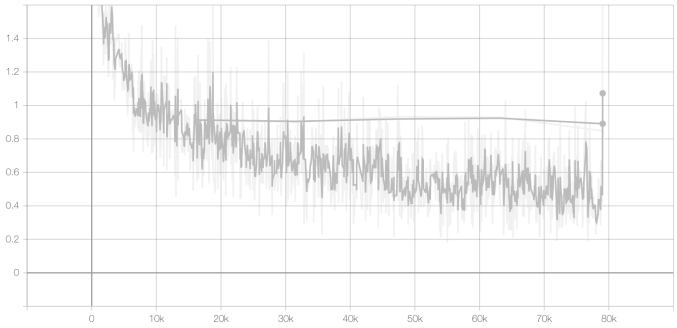


Fig. 7: MC Loss

IX. Qualitative Results

Both the LMC and MC correctly predicted 100648-1-1-0.wav, a car horn, and 100795-3-0-0.wav, a dog bark, which are both short, sharp and loud sounds.

MC was better at predicting engine idling sounds than LMC, with it correctly predicting 106014-5-0-1.wav, 106014-5-0-2.wav, and 106014-5-0-3.wav, with LMC getting them all wrong.

TSCNN was able to correctly predict 73524-0-0-6.wav, an air conditioner, and 27070-2-0-3.wav, children playing, where LMC and MC couldn't, showing the effectiveness of this late fusion method to classify complicated background sounds.

None of the methods were able to predict 118278-4-0-0.wav, drilling, another complicated sound with limited unique features.

X. Improvements

One improvement we wanted to implement was to replace the use of Stochastic Gradient Descent (SGD) with a more advanced method. We chose to focus on this optimisation after seeing from Fig.4-9 that we needed to much further reduce the loss in our model. As the loss is dependent on parameters such as the learning weights, we wanted to experiment with different gradient-based optimisation algorithms which take alternate approaches to updating the learning weights, hopefully improving the model accuracy.

We tested Adam [7], AdamW [8], and RMSProp [9]. In these adaptive learning rate methods, rather than maintaining a singular, unchanging, learning rate to update each weight as used in SGD, the learning rate for each network weight individually adapts as learning unfolds. Each of the separate optimisers offer their own improvements, such as applying smoothing parameters to further optimise the algorithms.

Through testing each of these optimisers, we discovered that AdamW provided the highest overall increase to accuracy, visible in Fig.10. AdamW is an extension of Adam that seperates the L2 regularization and weight decay steps, and has been shown to be better at generalization than Adam.

We can see from our results that both LMC and MC see an increase in overall accuracy, bringing our results for LMC into the 5% range of the target results, while MLMC ad TSCNN sees a decrease. The main decrease for all of our models is in class 1 (air conditioner), with TSCNN seeing the greatest decrease at 29%, which causes TSCNN's average accuracy to fall, despite its components (LMC and MC) seeing an overall increase in accuracy.

The accuracy and generalisation offered by AdamW makes it a suitable improvement to some of our models, if not a huge one.

XI. Conclusion and Future Work

The biggest challenge of our replicated study was identifying and overcoming the various inconsistencies within the original paper. This demonstrates the importance of using

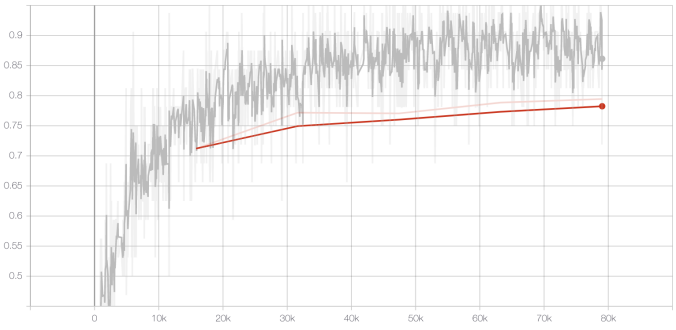


Fig. 8: MLMC Accuracy

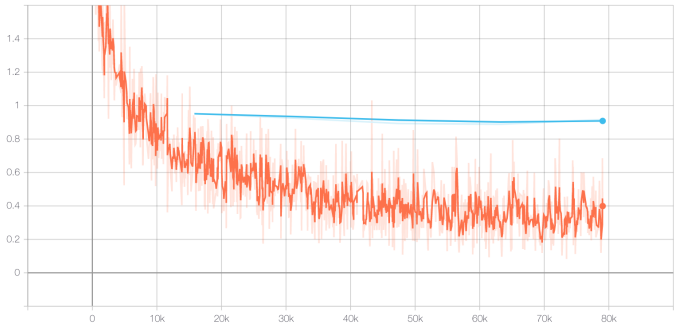


Fig. 9: MLMC Loss

detailed and clear diagrams within reports of this manner, and in taking caution about how architectures are conveyed. Once overcoming these issues and developing a working architecture, we were able to tweak details and parameters, such as the weight decay and epoch number, to increase the accuracy of our model towards the desired goal.

Our incorporation of the AdamW optimiser helped us marginally improve the accuracy of some of our models and meet the target results, and helped improve generalisation, but still did not offer a significant improvement to our models. In future works we would place more focus on improving the model accuracy by preventing over-fitting of the training network. It is evident from Fig.4-9, where our loss curves for training data is dramatically lower than our test data, that over fitting was an issue in our models.

A potential future work could be the introduction of data augmentation akin to that seen in Takahashi et al.'s work on combining sound files of the same class to create new data samples, combining the features of samples of the same class to create new data samples, and expand and add variation to the data set, helping to combat over-fitting.

Class	LMC	MC	MLMC	TSCNN
ac	35.0%	48.0%	43.2%	49.0%
ch	93.9%	81.8%	85.0%	90.9%
cp	84.8%	90.0%	82.0%	88.5%
db	80.0%	82.0%	95.0%	82.0%
dr	73.7%	81.8%	82.8%	79.9%
ei	76.3%	79.5%	64.5%	82.8%
gs	100%	99.0%	95.0%	100.0%
jh	100%	95.8%	99.0%	99.0%
si	60.2%	63.8%	65.0%	63.0%
sm	88.0%	85.0%	89.0%	87.6%
Avg	***79.2%***	***80.7%***	***80.0%***	***82.3%***
Change	*+0.6%*	*+1.8%*	*-1.1%*	*-0.8%*

Fig. 10: AdamW Results

REFERENCES

[1] Y. Su, K. Zhang, J. Wang and K. Madani, "Environment Sound Classification Using a Two-Stream Cnn Based on Decision-Level Fusion", Sensors (Basel, Switzerland). 2019.

[2] K.J. Piczak, "Environmental sound classification with convolutional neural networks", In Proceedings of the 2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (Boston, MA, USA). 2015.

[3] N. Takahashi, M. Gygli, B. Pfister, and L. Van Gool, "Deep Convolutional Neural Networks and Data Augmentation for Acoustic Event Detection", Sony Corporation (Japan). 2016.

[4] S. Chachada and C.-C. J. Kuo, "Environmental sound recognition: a survey", APSIPA Transactions on Signal and Information Processing, Cambridge University Press. 2014.

[5] J. Salamon, C. Jacoby and J. P. Bello, "A Dataset and Taxonomy for Urban Sound Research", 22nd ACM International Conference on Multimedia, (Orlando USA) Nov. 2014.

[6] Pytorch, "torch.nn — PyTorch master documentation", [online] Available at: https://pytorch.org/docs/stable/nn.html [Accessed 14 Jan. 2020].

[7] Kingma, D. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. [online] arXiv.org. Available at: https://arxiv.org/abs/1412.6980 [Accessed 15 Jan. 2020].

[8] Medium. (2018). Why AdamW matters. [online] Available at: https://towardsdatascience.com/why-adamw-matters-736223f31b5d [Accessed 15 Jan. 2020].

[9] Medium. (2018). Understanding RMSprop — faster neural network learning. [online] Available at: https://towardsdatascience.com/understanding-rmsprop-faster-neural-network-learning-62e116fcf29a [Accessed 15 Jan. 2020].