Reproducing and Improving a Two-Stream CNN for Environment Sound Classification

Ben Hermans
bh16979

Tim Roderick
tr16259

Ayshe Kuran
ak16531

I. Introduction

Deep learning-based models are a robust approach to tackling the problem of classification, where conventional methods have proven less so. An example of where these approaches have been applied is sound classification: trying to identify events in sound clips. A real world example of where this applied is in smart home technology. Auditory features such as log-mel (LM) spectrogram, mel frequency cepstral coefficient (MFCC) and raw audio waveforms can be used to train deep learning models designed to tackle this problem. In sound recognition, schemes often applied to one specific type of sound, such as music or speech, do not translate effectively to others, namely environment sound classification. This can be attributed to the lack of similar and repeated audio characteristics as found commonly in speech or music.

The approach taken by Su et al [1] is based around a stacked four-layer CNN architecture. Their first step is feature extraction, for which they extract five auditory features including LM and MFCC. After combining these, they train two CNNs separately, and use a late fusion method to combine these results. Through experiments they find that merging neural networks, trained on different features and using decision level fusion, gives better results than deep architectures trained on combined features in this classification problem.

In this paper we attempt to replicate the results achieved by Su et al [1] by reproducing their proposed method, specifically the four-layer architecture with a simplified late fusion approach and smaller train/test fold.

II. Related Work

In their work, Li et al [2] attempt to tackle the same problem of environmental event sound recognition. They use an ensemble stacked convolutional neural network. Both papers use the same method to fuse two CNNs for greater performance. The approach used by Li et al [2] involved training two CNNs, one with LM spectrograms as input, the other with raw waveform information. These two CNNs were combined using Dempster-Schafer (DS) evidence theory to form an ensemble DS-CNN model, just as the paper we are replicating. This was then tested on three datasets, with the UrbanSound8k being the common dataset across both approaches. Their CNN architecture was similar to that of Su et al [1], with the major change being an extra convolutional layer.

As an alternate approach to the problem of environmental sound classification, Salamon and Bello [3] explore the use of unsupervised feature learning. While this technique had previously been successfully implemented with music, they specifically take on the challenge for urban sound audio. They use a random forest classifier in their classification stage. This team also uses the UrbanSound8k dataset. With this method they manage to capture the irregular dynamics of environmental sound recordings. Some specific classes showed better performance, such as the engine idling and jackhammer classes, due to the presence of some short repeated audio characteristics.

III. Dataset

The dataset used is UrbanSound8k. This is a collection of 8732 urban sound clips (WAV files) that are less than 4 seconds long. The total length of all of these clips is about 9.7 hours. Each clip is labelled as belonging to one of 10 distinct classes: air conditioner, car horn, playing children, dog bark, drilling, engine idling, gun shot, jackhammer, siren, street music. The original paper used a 10-fold train/test split, however for speed we are using a simple train/test split of 9:1.

IV. Input

The LMC and MC inputs are aggregations of the auditory features chroma, spectral contrast and tonnetz, with the LM spectrogram and the MFCC features respectively. The LM spectrogram is a widely used feature in environmental sound classification, and it is a representation of the frequency spectrum on a log-mel scale, which is a modified frequency scale based on how people hear pitch. The feature used for MC on the other hand, MFCC, is the result of applying the discrete cosine transform on the LM spectrogram.

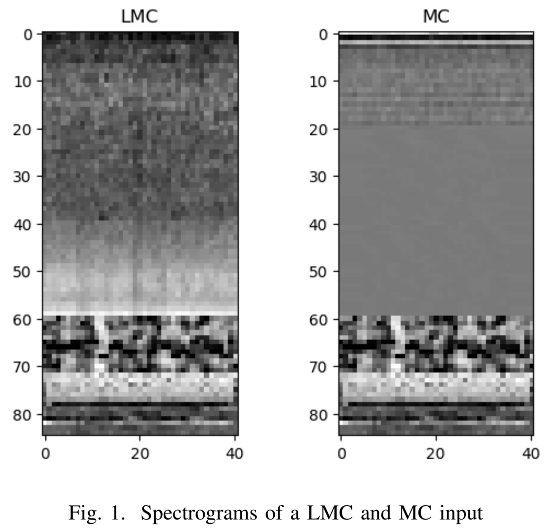


Fig. 1. Spectrograms of a LMC and MC input

In our implementation, these features are each of size 41×60 (as we set our LM feature to have 60 channels to match the size of MFCC). The sizes of the chroma, spectral contrast and tonnetz features are respectively 41×7, 41×6 and 41×12. After stacking the features with LM/MFCC on top of chroma, spectral contrast and tonnetz, in that order, the total size of the input is 41×85, giving us our LMC/MC inputs respectively. We will test our late fusion method against a network trained on the combination of both features; MLMC. The MLMC feature sets have size 41×145, and we simply stack the MFCC feature on top of the existing LMC input.

Exemplar LMC and MC spectrograms can be seen in Fig 1, where the four segments are clearly visible for some random test sample, where each segment has been scaled to be more prominent.

V. Architecture (Su et al [1])

We call the two networks based on the LMC and MC features LMCNet and MCNet respectively, and they both have the same 4-conv architecture, shown in Fig 2. In this diagram we label output sizes for each layer where the overall input size is 41×85. Each convolutional layer uses a 3×3 kernel size and 1×1 padding to retain the input size. The activation function used is ReLU [4]. We use 2×2 Max Pooling between the second and third convolutional layers also with the same padding (due to the odd dimensionality). The first fully connected layer uses the Sigmoid activation function. Finally, to improve generalisation of the networks, we use Batch Normalisation [5] on each of the convolutional layers (before the activation function) and standard Dropout [6] on the output of the second, fourth and fifth layers.

The description of the architecture presented in the original paper is inconsistent. Su et al [1] state that for their kernels in each convolutional layer, they use a stride of 2×2. If they indeed used a stride as suggested on each layer then the layer sizes would halve each time, whereas they document that they stay the same. This implies there are two solutions. Removing all the stride suggested and instead implementing another max pool layer between the convolutional layers and the first fully connected layer, or remove stride from all the convolutional layers except the final one. Both of these would result in the correct network size (and number of parameters, Fig 3) with only slightly different results. We opted with the latter of these options.

VI. Implementation Details

Once we had implemented the CNN architecture described in Fig 2, we attempted to replicate the LMCNet and MCNet results in the paper. To achieve the best possible results we needed to tweak variables which were not initially stated explicitly. The original paper seemed to suggest the use of the Adam optimiser [7], with learning rate 0.001, batch size 32, where we subject all weights to L_2 Regularisation and use a Dropout probability of 0.5. Cross entropy was used to compute loss throughout and Dropout was applied only in training. The original paper implied the use of a Softmax layer on the network output, but since PyTorch's implementation of Cross Entropy Loss does this automatically, we didn't explicitly include this. Similarly, they used a momentum of 0.9, however the Adam optimiser does not use momentum in the same way as typical Stochastic Gradient Descent (SGD), so this value is also surplus.

Upon initial creation of the network from Su et al [1] we made sure that the number of parameters in the model was correct, using a PyTorch Summary script to produce the table shown in Fig 3. Originally our number of parameters was too high, the difference being the "bias" values for each layer which we subsequently removed to get the right figures. Note that in Fig 3 it displays Dropout2D layers, however we ultimately changed these to standard Dropout layers as it gave better performance despite the first two Dropout layers acting on 2D data, where these more complicated Dropout layers should benefit generalisation. Similarly, the position of the Dropout layers was ambiguous in the paper. After trying different possible positions, such as on the input of the layer or before the activation function, we decided to have the Dropout on the output as described, as this is most conventional.

We found, upon implementing the described model, that it was prone to significant overfitting, which is noticeable in Figures 4 and 5, despite the Batch Normalisation, Dropout and L_2 Regularisation techniques included to mitigate this. The network seemed to learn incredibly quickly, reaching a local minimum within a few epochs with the validation accuracy fluctuating consistently afterwards. We used the

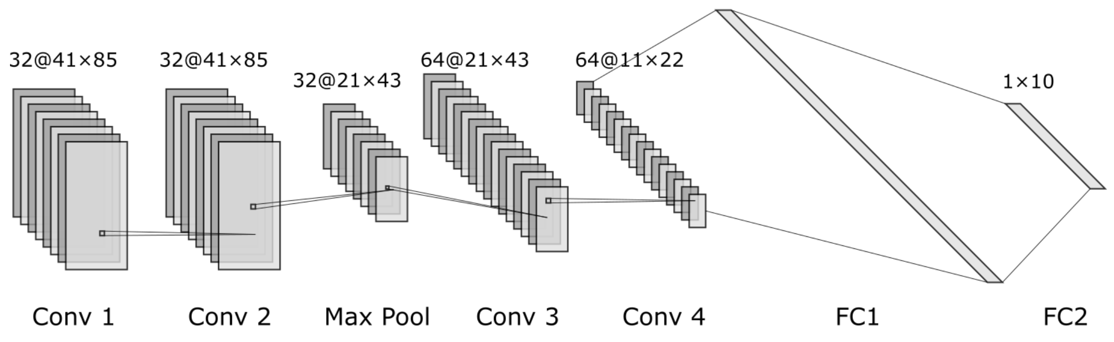
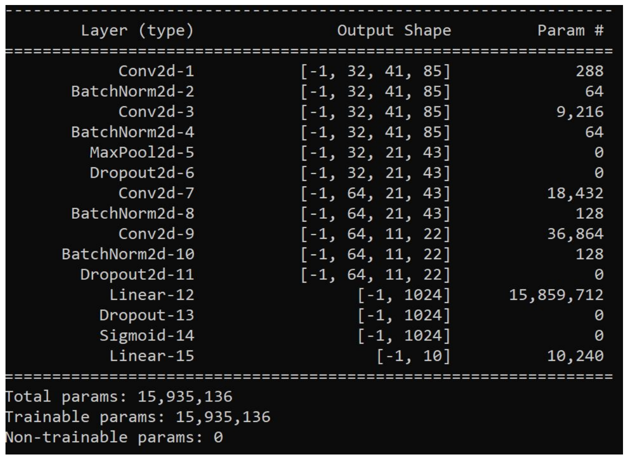


Fig. 2. Diagram of the DNN we reproduced with labelled output sizes



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 41, 85]             288
       BatchNorm2d-2          [-1, 32, 41, 85]              64
            Conv2d-3          [-1, 32, 41, 85]           9,216
       BatchNorm2d-4          [-1, 32, 41, 85]              64
         MaxPool2d-5          [-1, 32, 21, 43]               0
         Dropout2d-6          [-1, 32, 21, 43]               0
            Conv2d-7          [-1, 64, 21, 43]          18,432
       BatchNorm2d-8          [-1, 64, 21, 43]             128
            Conv2d-9          [-1, 64, 11, 22]          36,864
      BatchNorm2d-10          [-1, 64, 11, 22]             128
        Dropout2d-11          [-1, 64, 11, 22]               0
           Linear-12                [-1, 1024]      15,859,712
          Dropout-13                [-1, 1024]               0
          Sigmoid-14                [-1, 1024]               0
           Linear-15                  [-1, 10]          10,240
================================================================
Total params: 15,935,136
Trainable params: 15,935,136
Non-trainable params: 0
```

Fig. 3. A breakdown of the number of parameters in our CNN

amsgrad option for the Adam optimiser to mitigate unwanted convergence [8]. In PyTorch, L_2 regularisation is implemented by a weight decay term, which we set to 0.00001 to generate our results. Research suggests that performing this regularisation on Adam is ineffective unlike typical SGD and instead a proper implementation of weight decay, rather than L_2 regularisation, should be used - implemented in the AdamW optimiser [9]. Decreasing the learning rate or using SGD with Nesterov Momentum for optimisation provided slower learning but ultimately the same overfitting occurred, oscillating around similar values. Ultimately we decided to stick with the standard Adam optimiser as to stay true to the original paper despite the theoretical improvements to overfitting other configurations can provide. We trained both networks for 50 epochs, shuffling training data each time.

To evaluate the performance of the networks prior to fusion, we needed to group audio segments. Each audio clip is split into a number of segments during training and for testing, however to properly evaluate whether an audio clip was correctly classified we need to average the predictions on each of its segments. These predictions are then what is used to compute the validation accuracy seen in Fig 5 and in our results table, Fig 4. We used a simple train/test fold instead of the original 10-fold approach meaning we may have more bias or optimistic results. We then saved the combined predictions (logits vector) for each audio file into a JSON file via an internal dictionary. This would happen in every validation call during training as well as after-the-fact using the trained network (in form of a checkpoint file, .pt) using our test script.

Once we had successfully trained LMCNet and MCNet and replicated the results required, we implemented a simplified late fusion method. Saving the results of each network as a JSON allowed us to combine the results immediately. Creating a list of correct/incorrect predictions (overall and per class) for each network was simple by averaging the logits between the two networks for the same audio files. This produced our combined prediction, TSCNN.

Class	LMCNet	MCNet	MLMC	TSCNN
aircon	50.0%	48.0%	36.0%	47.0%
carhorn	93.9%	90.9%	90.9%	90.9%
children	89.0%	91.0%	83.0%	91.0%
dog bark	80.0%	81.0%	78.0%	81.0%
drilling	80.0%	79.0%	81.0%	80.0%
engine	56.9%	65.6%	61.3%	62.4%
gunshot	96.9%	96.9%	96.9%	96.9%
jackhammer	100.0%	92.7%	97.9%	98.9%
siren	59.0%	63.9%	54.2%	66.3%
street music	94.0%	80.0%	84.0%	90.0%
Average:	79.9%	78.2%	76.3%	80.4%

Fig. 4. The Per Class Accuracy results for replicated networks and late fusion

VII. Replicating Quantitative Results

Our replicated results can be seen in Fig 4. We expected better performance on our LMC network and worse on our MC network but we believe the difference is down to the nature of the features. Despite this the results have been replicated quite closely.

VIII. Training Curves

The training and validation curves for the LMC, MC and MLMC networks can be seen in Fig 5. In each of the training graphs we have the expected result: training accuracy increases while loss decreases. It does seem that the networks learn too fast, primarily due to the combination of the Adam optimiser and the relatively high initial learning rate stated in the paper. The validation curves are more worrying, as validation accuracy fluctuates around a steady rate with very little initial growth. This is down to a combination of this quick initial growth and never improving on the local minima that was found. The validation loss curves actually show increasing loss over time for our validation set which is a large indicator of overfitting in our network. The loss represents how confident in the predictions the network is and over time, as the network becomes increasingly specialised to the training set, the outputs for our validation set become less and less sure.

Overfitting is clearly a major problem with this network despite all the precautions proposed in the original paper being implemented. It seems as though for a network of this size we have not got enough data to generalise well, where perhaps a smaller network like that suggested by Li et al [2] is better suited for the dataset size. The speed of our network makes it seem like we barely learn at all, although we do, just incredibly quickly.

IX. Qualitative Details

To provide some qualitative analysis of our network performance we made our fusion script output sets of matrices corresponding to the indices of sound clips that met certain criteria. Example inputs for the first three of these cases can be seen in Fig 6. There were no cases where both the LMCNet and MCNet were unsuccessful but the fusion method correctly

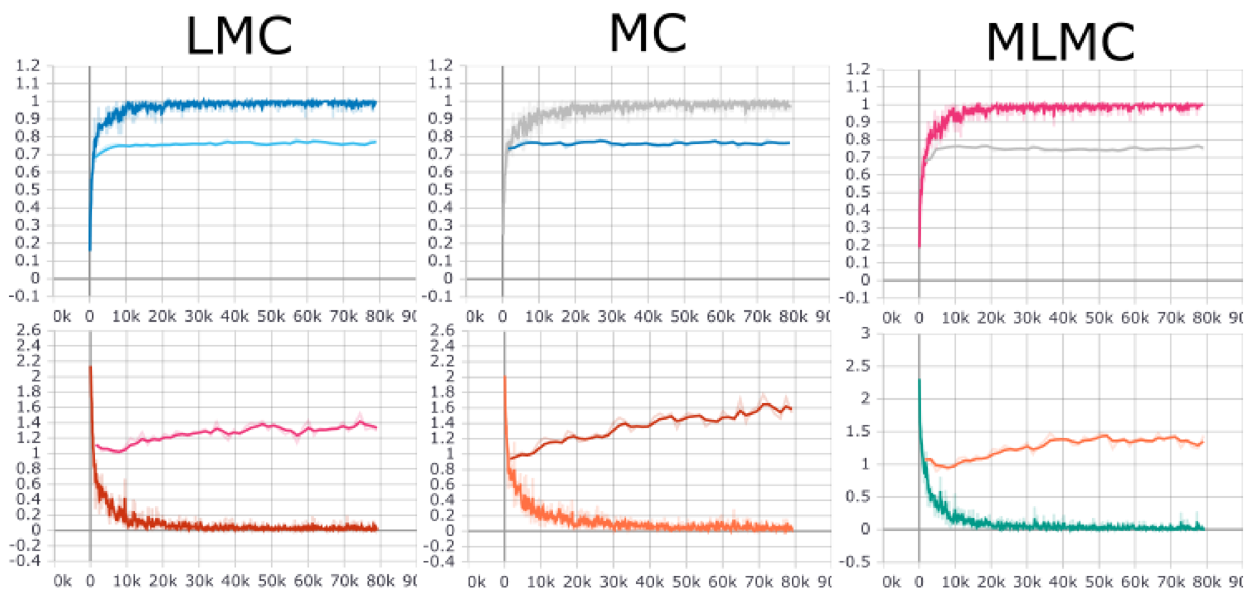


Fig. 5. Training/Validation Accuracy (top) and Loss (bottom) curves for LMC, MC and MLMC

classified, showing that the fusion is entirely dependent on the performance of the original networks: if these suffered due to overfitting then the fused results are less likely to improve in areas both networks were wrong, as the predictions were already too specialised.

In Fig 6 the first input, 50, was incorrectly classified by both networks. Looking at the relevant output vectors of each network one can see that they both predicted, incorrectly, that the event was a jackhammer as opposed to an engine. Based on our results this is expected as both networks reported a success of $> 90\%$ for jackhammer classification but $< 66\%$ successful for engines - signalling a specialisation toward jackhammers detracting from the engine classification. Also depending on the clarity of the clips these are very similar sounds - repetitive and harsh, which may contribute to similar initial feature sets. Noticeably in this example the MC network is actually not too wrong, $2/3$ sure about jackhammer and $1/3$ toward engine. However the LMC network was significantly worse with almost no weighting attached to the correct label, perhaps due to the lack of noticeable patterns in this particular feature space.

In the same figure, the input indexed by 15 was correctly classified (children), whereas 16 was only classified correctly by one despite being of the same class. The LMC network was very sure about 15 and was only just correct about 16 - about 60/40 between children or dog barking. MCNet on the other hand was incredibly sure about 16 being a barking dog as opposed to children. This could of course depend on the type of dog - perhaps a higher pitched dog bark could sound like children. This was one of the examples where the fusion did not predict correctly as the MCNet prediction was so *certain* whereas the LMC was not, despite being correct. This demonstrates the weakness of this simple fusion method, succumbing to the specialisation of one network despite the correctness of another.

Finally, Fig 1 shows an input that was only correctly classified by MCNet, which was confident in it's prediction. LMCNet's prediction was rather general, about 45/35 between two incorrect labels and small weightings for the others. Luckily, due to the certainty (and that LMCNet was not dramatically uncertain) of MCNet the fusion method combined the results and classified the input correctly. This sample was an engine sound and clearly from Fig 4 our LMCNet is much less accurate than MCNet for this sound event, but the 10% more accuracy in MCNet makes up for the other network. In 165 cases where one prediction was correct and the other was not, 58 were correctly classified by fusion leading to the improvement shown by TSCNN over LMCNet and MCNet.

Based on Fig 4 and looking at the examples which weren't classified correctly, it is clear that this simple late fusion method brings the best out of both individual networks. However it is also clear it does not do anything to favour the samples where both networks were wrong.

X. Improvements

Looking at our graphs in Fig 5 we felt that we should be able to tweak the hyper-parameters of our model to reduce the training accuracy slightly in return for better test accuracy. We initially experimented with tweaking the learning rate, with lower rates giving us a much smoother curve but ultimately no increase in performance. Instead, we varied the Dropout rate. Although 0.5 is often the best Dropout probability for retaining information in hidden layers, values closer to 1 perform better at retaining information from the initial inputs [6]. We tried a range of values between 0.5 and 0.9 and found that running the model with a probability of 0.7 found a sweet spot between better generalisation (and thus worse accuracy) and what we had already achieved. We found the natural range of accuracies produced with this change averaged a couple percent higher than before, and the results of a single run under this changed parameter can be seen in Fig 7.

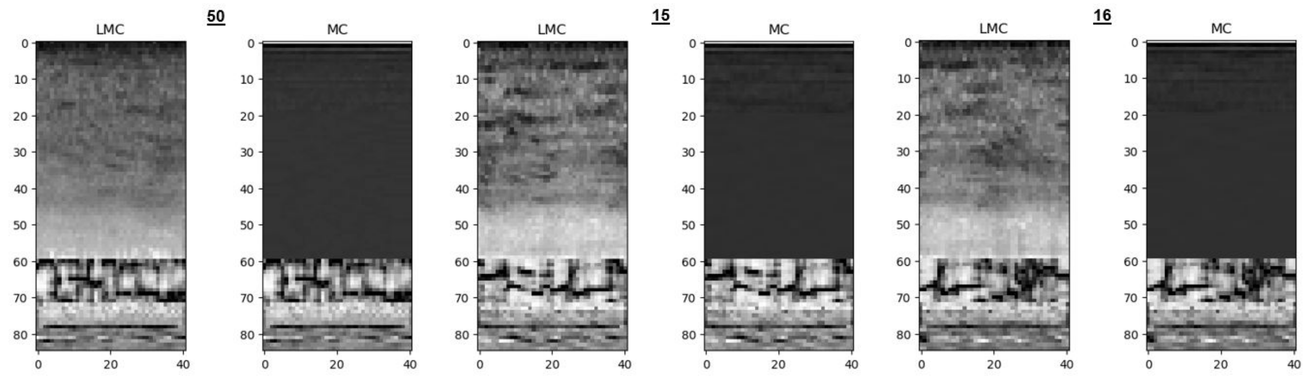


Fig. 6. Test Inputs which were correctly classified by neither networks, both network and one of the two networks (indexes 50,15,16 respectively)

We explored further ways we could improve the model's capability, particularly improve the overfitting. We experimented with many ways to do this, using more impactful weight decay [9] or adding more Dropout/batch normalisation layers. Another way to improve generalisation is to train on more data, so we tried data augmentation: Gaussian Noise. We implemented a custom Gaussian noise layer and parameter but it did not improve our results. Finally we experimented using SGDR [10], Cosine Annealing to control the learning rate for standard SGD optimisation with Nesterov Momentum where the learning rate is reset regularly to deal with settling in local minima. All of these approaches suffered the same fate, they could all be run to improve generalisation but in expense of some accuracy which is completely expected. One method we would have tried with more resources is Early Stopping, as it was clear in many of the training phases we found that after reaching a reasonable accuracy it would just fluctuate with validation loss increasing. In many of our runs, including using a higher Dropout, effective Early Stopping would have improved our accuracy further as we would have stopped training at the best time.

XI. Conclusion and Future Work

In this report we have shown that we were able to reproduce the results of Su et al [1] to a reasonable degree of accuracy using the model they described. In review of this model it is clear that the model suffers from overfitting despite the mitigations put in place. We have shown how difficult it is to replicate published papers which don't give enough explicit information or provide sample code, instead having to try different configurations to try and achieve a similar result. We highlighted many areas in the original paper where we found inconsistencies as well as theoretical shortcomings. We explored several improvements to try and reduce this further but ultimately, as expected, could not increase the overall accuracy while doing so, instead increasing generalisation and showing a truer representation of model. We did of course manage to improve on the model by fine-tuning the Dropout probability, although there is definitely room for further experimentation, possibly with a completely new architecture. We would suggest a new model built for this task would incorporate many of the techniques we experimented with in our improvements. Ultimately, we are pleased with the degree which we were able to reproduce the model given and would be interested in what a better performing architecture may be.

Class	LMCNet	MCNet	MLMC	TSCNN
aircon	49.0%	69.0%	48.0%	63.0%
carhorn	93.9%	75.8%	96.9%	90.9%
children	92.0%	95.0%	80.0%	95.0%
dog bark	74.0%	72.0%	81%	75.0%
drilling	79.0%	83.0%	79%	82.0%
engine	70.9%	80.6%	79.6%	78.5%
gunshot	100.0%	100%	96.9%	100.0%
jackhammer	98.9%	91.7%	98.9%	98.9%
siren	61.4%	56.6%	56.6%	61.4%
street music	93.0%	86.0%	88.0%	90.0%
Average:	81.2%	80.9%	80.5%	83.5%

Fig. 7. Improved Per Class Accuracy results for our networks and late fusion

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*, 19(7):1733, 2019.

[2] Shaobo Li, Yong Yao, Jie Hu, Guokai Liu, Xuemei Yao, and Jianjun Hu. An ensemble stacked convolutional neural network model for environmental event sound recognition. *Applied Sciences*, 8(7):1152, 2018.

[3] Justin Salamon and Juan Pablo Bello. Unsupervised feature learning for urban sound classification. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 171–175. IEEE, 2015.

[4] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[5] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[6] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[7] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[8] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. On the convergence of adam and beyond. *arXiv preprint arXiv:1904.09237*, 2019.

[9] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2018.

[10] Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.