Applied Deep Learning

Daniel Davies (35349)
dd16785

Chetankumar Mistry (38523)
cm16161

I. Introduction

As is the beginning of many deep learning papers, this paper tries to improve the automation of another human capability; this time, in the recognition of environmental sounds. The field altogether is known as Intelligent Sound Recognition (ISR), a pivotal part of which is ESC - environmental sound classification - that is, assigning classes to some detected sounds. Alongside ESC sits (loosely) two classes - human voice (ASR) and musical sounds (MIR) - of which ESC is the broadest category. ASR and MIR sounds have been fairly well researched, with significant developments in both; Su et al claim that, unfortunately, the research here is not efficiently transferable to the environmental sound classification problem due to it's broadness, and hence, this paper aims to address this knowledge gap in ISR. This is significant as a precise/accurate ISR could affect many applications, such as aiding clinicians in healthcare, to even rainforest conservation through acoustic surveillance [1]. The paper identifies two main obstacles for environmental sound classification to date:

- Currently the features used in classification of ESC audio (features being frames of the audio, filtered by certain techniques) are being used individually to classify audio. The researchers believe this fails to capture important information in audio, and that a way of combining these meaningfully is needed.
- The problem also needs a new architectural solution, namely, an application of deep learning on this new formulation of data. They claim current models (such as SVMs) are not designed for such a task, and hence CNNs (which can model more complexity of data) need to be explored as a more powerful alternative. Not only this, but the researchers hope for promising results in using multiple CNNs together in an ensemble modelling fashion as output, which they hope will capture properties such as longer temporal contextual information in the audios.

Together, the researchers hope these will outperform current methods for environmental sound classification.

II. Related Work

Due to the increasing number of potential applications of sound classification, there have been numerous alternative approaches to resolve this issue through the use of CNNs. One such example, similar to what we present, is by Boddapati et al. [2] who took audio snippets, provided by the ESC-10, ESC-50 and UrbanSound8k datasets, and extracted certain features, namely the Mel-Frequency Cepstral Coefficients (MFCC), Spectrograms and Cross Recurrence Plots, treating them as black and white images and passing them into the AlexNet and GoogLeNet image recognition architectures. Not only this but they propose combining these features into a single RGB image where each auditory feature corresponds to a single channel and then pass these images to the CNNs. Their results show that GoogLeNet performed better than AlexNet in almost all cases, and that the combination of 3 features into an RGB image provided significantly better results than individually.
In 2015, Piczak [3] made the claim that sound recognition with CNNs is troublesome due to the lack of data available. He too used the 3 publicly available datasets (ESC-10/50, UrbanSound8k), extracting different audio features from each sample, and proposed an interesting CNN architecture, which composed of 2 convolutional layers, followed by 2 fully connected layers. What is interesting about his architecture, is that when compared with others he uses rectangular shaped filters for his convolutional and pooling layers rather than square, with the first convolutional filter being of shape 57x6, the second being 1x3, and for his pooling layers he used shape 4x3 and a stride of 1x3 for his first layer, followed by a shape of 1x3 and stride of 1x3 for the second pooling layer. Using 5-fold cross validation, his architecture yielded low results by today's standards with the peak accuracy being 64.5% for the ESC-50 dataset, and 73.7% for UrbanSound8k. However, at the time, these results were significantly better than their respective, manually-engineered features.
Salamon and Bello [4] took the idea proposed by Piczak [3], and continued upon it, by partially solving the issue raised by Piczak - lack of data available - by augmenting the data that they had at hand. Using the UrbanSound8k dataset, they proposed a CNN architecture to learn spectro-temporal patterns that are representative of each sound class, and, like the others, used extracted audio features to pass into their CNN. Their proposed CNN architecture consists of three convolutional layers interleaved with two pooling layers, followed by two fully connected layers, utilising the ReLU activation function for all layers except the last where they use Softmax. They compare their architecture's results with the best performing architecture proposed by Piczak, as well as a dictionary learning approach. Their results for unaugmented data was comparible with the other implementations, however, upon applying the augmentations to increase their dataset size, their mean accuracy's increased by 5%. The augmentations they applied were: Time Stretching, Pitch Stretching, Dynamic Range Compression, and Background

noise, and they claimed that pitch shifting yielded the greatest improvements in accuracy, whilst certain classes, such as ”air conditioner”, suffer from the Background Noise augmentation due to the fact that the class is characterised by a persistent ”hum” sound.
By looking at these alternative works, it is clear that the problem of sound classification is still ongoing, and there currently is no perfect solution. One key problem we notice is the distinct lack of data available, which leads us to believe that the best way to improve results given this handicap, is to be creative with the data we do have access to. Whether it is augmenting data, combining data into RGB images, or other interesting ways, resolving this will be effective in improving the accuracy’s and results that any CNN will yield.

III. Dataset

The dataset used in this paper is a common ESC dataset, used in many of the related works, namely the “UrbanSound8K” dataset. This contains 8732 audio files (of 4 seconds or less), of urban sounds, which researchers initially obtained by creation of a taxonomy of sounds presented in the original paper [5]. Researchers focussed on the low levels of the taxonomy for this. The 10 specific classes were selected as the most “useful” by the researchers since they are the most frequently reported noise complaints in NYC (except for ”Gunshot” and ”Children playing”, which were added for diversity [5]). This dataset is labelled (manually) with the 10 individual classes (specifically, air conditioner (ac), car horn (ch), children playing (cp), dog bark (db), drilling (dr), engine idling (ei), gun shot (gs), jackhammer (jh), siren (si) and street music (sm)).

Originally, each of the files are given as (links to) WAV files, for the user to apply their own formatting to. In this paper, each WAV file is first split up into 41 frames with a cosine window function, each of which then has several filters applied to obtain the individual categories that are used for classification (’logmelspec’, ’mfcc’, ’chroma’, ’spectral_contrast’ and ’tonnetz’). These filters essentially apply things like Fourier transforms, or study pitches of the audio etc, but most importantly, are implemented in libraries that the researchers have used. Each of these filters will produce a variable number of channels; 60 for logmelspec and mfcc, and 7, 12 and 6 for , ’chroma’, ’spectral_contrast’ and ’tonnetz’.

Fortunately for us, the dataset was provided preprocessed; the stages of splitting an audio into frames, and then extracting each of the audio features had been completed, and the data was given as a simple pickle [6] file (corresponding to a dictionary of the frame samples- each sample already containing the filename and class, as well as a features dictionary, containing the individual features.

Along with the WAV files, the researchers who created the data set also provide a group number for each of the files, which have been predetermined as the best group categorisations to evaluate your model. There are 10 group numbers, which can be used to perform 10 fold cross validation on, and indeed this is what the original paper does. In our case however we have already been given a predetermined train-test split (“single fold validation”). On each epoch, the data loader of the training data shuffles the data, while the testing data loader keeps the order the same throughout.

IV. Input

When we have the individual features available, we are able to produce ”aggregated” features that the researchers claim give a better performance. These are:

First, the researchers define ’CST’ as the linear combination of ’chroma’, ’spectral_contrast’ and ’tonnetz’

- LMC: This is the name given to the mixture of ’logmelspec’ and CST
- MC: This is the name given to ’MFCC’ and CST

The audio is split into 41 frames, and as mentioned above, each of these filters has their own size. If we combine the features linearly, and add the sizes, then CST has a height of 25, and ’MFCC’ / ’logmelspec’ both have a height of 60. Hence the total shape of any of the audio segments for either LMCNet or MCNet is 41 x 85. We can plot these as images to obtain the following (example taken from training set image index 78):

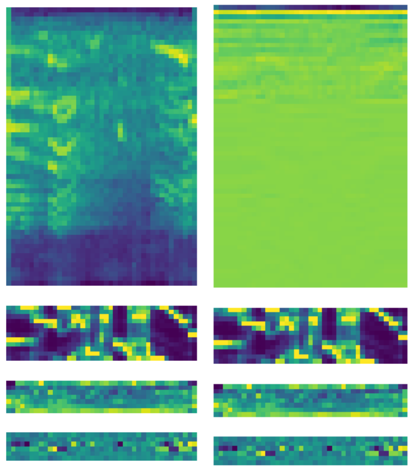

(a) LMC (b) MC

Fig. 1: Images that are used by LMCNet and MCNet respectively

V. Architecture

A summary of the architecture is displayed in the below tables. Table 1 will present the convolutional part of the architecture, and table 2 will show the fully connected layer descriptions. Both focus on input sizes for LMC and MC (but MLMC follows the same principle).

Looking at the tables, there are two inconsistencies that can be seen:

Layer	Kernel size	Stride	Batch Norm?	Dropout?	Im Size
Conv 1	3x3	2x2	Yes	No	41x85
Conv 2	3x3	2x2	Yes	Yes	41x85
Pooling	2x2	NA	NA	NA	41x85
Conv 3	3x3	2x2	Yes	No	21x43
Conv 4	3x3	2x2	Yes	Yes	21x43

TABLE I: Su et Al's Convolutions

Layer	Input Size	Output Size	Dropout?
FC	1024	10	Yes

TABLE II: Su et Al's Fully Connected Layer

Layer	Kern.	Stride	Bat.Norm?	dropout	Size
Conv 1	3x3	1x1	Yes	No	41x85
Conv 2	3x3	1x1	Yes	Yes	41x85
Pooling	2x2	2x2	NA	NA	41x85
Conv 3	3x3	1x1	Yes	No	21x43
Conv 4	3x3	2x2	Yes	Yes	21x43

TABLE III: Our Convolutions

Layer	Input Size	Output Size	Dropout?
FC	15488	1024	No
FC	1024	10	Yes

TABLE IV: Our Fully Connected Layers

- Stride: the paper claims that each of the convolution layers have a 2x2 stride associated to them. Assuming they meant stride in the traditional sense, this cannot be the case; if it were, the image size would halve in each dimension after every convolution layer, or have an impractically large padding (half the image size). Hence in our proposed model, we remove stride from all convolution layers completely, apart from the last one, which reduces the image size to 11x22.
- Fully Connected Layer size: this isn't so much an inconsistency, but information being left out. The researchers have stated that the (hidden) layer to the fully connected part of the network has 1024 units, mapping to 10. Hidden layer implies another FC layer above this one, to use the 15488 data points remaining after the last convolution layer (64 lots of 11x22 images). Hence, we added another fully connected layer which would map these 15488 to the desired 1024.

VI. Implementation

Our first step was to understand and implement the Su et al. architecture as described in their paper. This meant understanding the table of values displaying the parameters for each of the convolutional layers. All of the values were coherent except for the number of parameters going into the first FC layer, which was 15.9M. We realised that this was equal to $11 * 22 * 64 * 1024$ where 64 is the number of output channels from the previous layer, 1024 is the output of this FC layer and 11 and 22 is half of the image size being outputted from the 4^{th} convolutional layer. Once we understood how each layer feeds data to the next, we could go about implementing the architecture, achieving this by taking inspiration from publicly available Pytorch implementations of AlexNet [7] and then using the convolutional layers described there, changing them to be consistent with the architecture described by Su et al.
Tables 3&4 shows our proposed architecture and we will explain some of key differences between Su et al. and us here.

The main noticeable difference between our architecture and Su et al. is that we use a stride of 1x1 for all layers except the pooling and final convolutional layers, allowing us to use a small padding of 1x1.

A. Replication of Results

Upon implementing the architecture proposed by Su et al. we noticed that our performance was significantly lower than it needed to be, thus we analysed our architecture to see what appeared correct, what didn't, and what could be tweaked to improve performance.
One of the first things we noted was the location of dropout. As stated above, Su et al. are ambiguous as to where exactly the CNN applies dropout; we initially applied dropout solely on the first FC layer, and then through experimenting, evaluating results and performing research we found that the best position for the Dropouts were after the 2^{nd} convolutional layer, after the 4^{th} convolutional layer, and after the first FC layer. We felt this is justified as the paper stated that the dropout was "at" these layers, and we found no significant difference in results when placing them before or after the convolutions themselves. We then turned towards hyperparameters for performance gains. We noted that there was only really one parameter in our optimiser (Adam version of SGD) that could be tweaked to vary our performance:

- Weight Decay (L2 Regularisation)

The reason for this is that Su et al. clearly defined other hyperparameters such as Learning Rate at 0.001, Batch Size at 32, Dropout probability at 0.5 and Momentum at 0.9. Our target therefore, was to settle on a satisfactory value of weight decay since Su et al. mention that they subject all weight parameters to L2 regularisation, which is implemented as weight-decay in PyTorch. This was done by exploring other known architectures to see what an appropriate initial value should be, and then experimenting with that value and deviations from it. We finally decided that a value of 0.0001 gave us the best performance.

VII. Results

We ran our models multiple times, and found them to be quite variant: Across 40 runs LMC for example could range from 76% accuracy to 82%. However below we produce

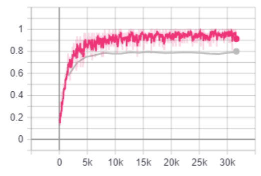
Class	LMC	MC	TSCNN	MLMC
ac	60.00	47.00	45.00	0
ch	90.91	84.85	90.91	0
cp	84.00	80.00	95.00	0
db	76.00	85.00	79.00	0
dr	85.00	69.00	79.00	0
ei	58.06	75.27	67.74	0
gs	100.00	93.75	100.00	0
jh	100.00	81.25	100.00	0
si	59.04	50.60	57.83	0
sm	87.00	87.00	86.00	0
Average	80.00	75.37	80.05	76.99

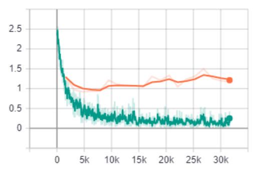
TABLE V: Our Results

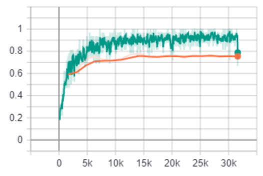
what we believe to be representative versions of our table 2 (above):

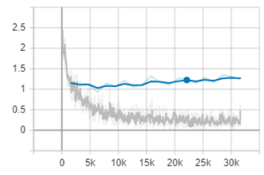
VIII. Accuracy and Loss curves

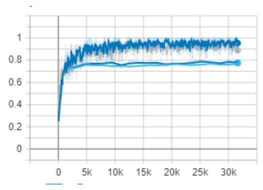
Corresponding to the above table of results, below are our accuracy and loss curves, for each model, taken from tensorboard:

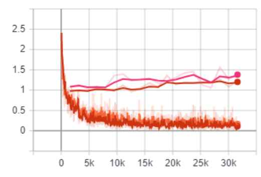

(a) LMC Accuracy

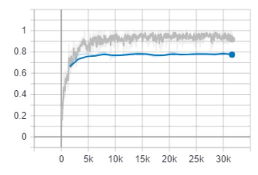

(b) LMC Loss

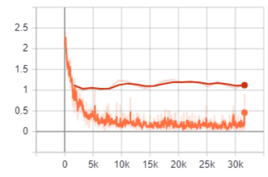

(c) MC Accuracy


(d) MC Loss


(e) TSCNN Accuracy


(f) TSCNN Loss


(g) MLMC Accuracy


(h) MLMC Loss

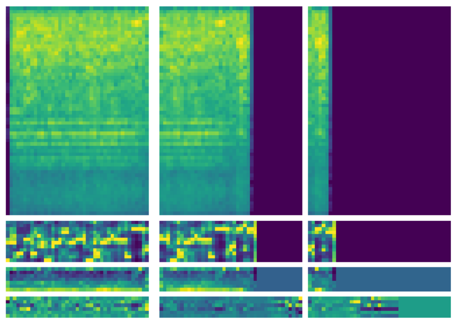
A. Overfitting

Our model is currently over-fitting the training set; when running our experiements, while batch accuracy remained at roughly 90-100%, our model consistently stayed at 75-80%. This can be seen in virtually all of the accuracy and loss curves. This was the best we could do with the given time limit and constraints from the paper, but in future it could have been useful to experiment with different probability of loss on dropout, or having more extensive look at parameters such as regularization.

IX. Qualitative Results

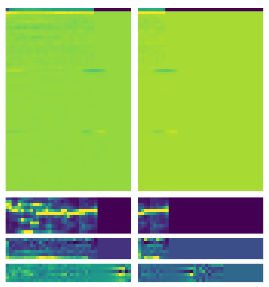
Audios are classified on a file basis rather than a segment basis, and so an audio file will have multiple segments. To show the audios, we have provided the segments in order, horizontally.

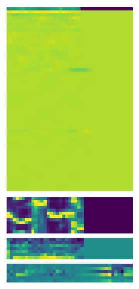
Below is the audio segment (given as the logmelspec image) for which both LMC and MC predicted correctly:


(a) 100648-1-1-0.wav

Interestingly, we can see that the audio file runs out to the end of the middle frame and then re-appears in the third frame. This is because the audio segments overlap.

Below, we see two charts: (a) shows the mfcc image for the case where MCNet correctly predicts something that LMC fails on, and (b) shows the output of the TSCNN, which correctly classified something that LMC did not:

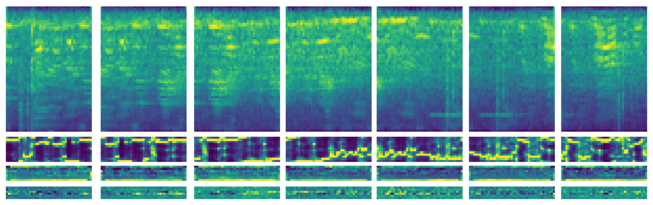

(a) MCNet Successful 100648-1-0-0.wav


(b) Late Fusion Successful 102103-3-0-0.wav

Finally, we see a seven segment image for which all models fail.

X. Improvements

To improve the performance of our system, we draw inspiration from Boddapati et al. [2] who generated an RGB



(a) 101382-2-0-45.wav

Class	RGB Image
ac	49.00
ch	100.00
cp	90.00
db	72.00
dr	80.00
ei	73.12
gs	93.75
jh	97.92
si	54.22
sm	88.00
Average	79.80

TABLE VI: Our Improved Results

image from extracted auditory features. We compose our image from the Log-Mel Spectrogram (R), Mel Frequency Cepstral Coefficients (G), and the concatenation of Chroma, Spectral Constrast and Tonnetz, padded with 0's so that the dimensions match. We do this, since - as demonstrated by Boddapati et al. - combining features into an RGB image and then passing that into an image classifying architecture can yield a high performing classifier. Since the architecture provided by Su et al. is also an image classifier, we can use an almost unmodified version of our architecture, with the majority of work affecting the dataloader. The changes needed to the architecture are as follows:

- Number of In-Channels for the first Convolutional layer
- The number of In Features for the first Fully Connected Layer

The first change is trivial; originally would would concatenate our respective audio features on top of one another in a single dimension, this meant that there was only a single channel to our model. Since we are instead stacking the features, we need to change the number of input channels our model takes to represent this, and so we set it to 3, which corresponds to the the R, G and B channels.
The second change is due to the fact that our image has reduced dimensions when compared to the original input. Since we are no longer concatenating the data, the dimensions of the image have been reduced, and so the first FC layer needs to be updated to acknowledge this.

XI. Conclusion and Future Work

In conclusion, we have shown a simple reproduction of the results in Table 2, inspired by Su et al in the problem of Environmental Sound Classification. Additionally to the basic problem, we have presented an extension in which we have shown that interpretation of MLMC data as RGB colours increases average accuracy by about 4.43% to MCNet. For the remaining tasks, we were able to achieve within 5% of the quotes in table 2 in general, although our model does vary slightly above and below this.

Any significant improvements on accuracy would however be seen by following the original paper more closely; namely, implementing the 10 fold cross validation, which would artificially increase our training and tests sets to help reduce over-fitting. Another major improvement could be the additional convolutional layers as presented in Su et al, which could (with a correspondingly larger data set) enable us to model a greater variety of detail/complexity in the audios.

References

[1] J. L. B. Y. Z. N. J. L. Yuan LiuHuawei, Zhongwei Cheng, "Ai for earth: Rainforest conservation by acoustic surveillance."

[2] J. R. L. L. Venkatesh Boddapatia, Andrej Petefb, "Classifying environmental sounds using image recognition networks," tech. rep., Blekinge Institute of Technolog, 371 79 Karlskrona, Sweden, 2017.

[3] K. J. Piczak, "Environmental sound classificationwith convolutional neural networks," tech. rep., Institute of Electronic SystemsWarsaw University of Technology, 2015.

[4] J. P. B. Justin Salamon, "Deep convolutional neural networks and dataaugmentation for environmentalsound classification," tech. rep., March 2017.

[5] J. P. B. Justin Salamon, Christopher Jacoby, "A dataset and taxonomy for urban sound research," tech. rep., Center for Urban Science and Progress,New York University.

[6] "pickle - python object serialization." https://docs.python.org/3/library/pickle.html.

[7] "alexnet.py." https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py.