Recreating Results from Valenti et al DCASE 2016 Acoustic Scene Classification

Dan Watkinson
Department of Computer Science
University of Bristol
Bristol BS8 1UB, U.K.
ky18001@bristol.ac.uk

***Abstract*—This report presents our work on the replication of published work by Valenti et al in solving the DCASE 2016 challenge of acoustic scene classification using a convolutional neural network. We achieve an accuracy of 86.4% when running full training on the model. The work details the architecture of the network and contains information on the implementation using PyTorch. We also propose an improvement to the system in an attempt to increase accuracy.**

***Index Terms*—convectional neural networks, dcase, acoustic scene classification.**

I. Introduction

Acoustic scene classification (ASC) is the capability of a human or artificial system to understand an audio context from an audio recording or stream. Humans use the concepts of ”context” or ”scene” to identify a given sound environment. For example humans can identify an audio scenario such as a coffee shop by listening to the ensemble of background noises and sounds in a recording. This recognition that seems like a simple task involves many complex calculations in the brain, drawing from many past experiences but allows us to easily associate these background noises to specific real-life scenes. Although simple for humans, this task is not easy for artificial systems. ASC can be useful in computation to support context aware computation [1], intelligent wearable devices [2] and mobile robot navigation [3].

The work of Valenti et al [4] applies the use of convolutional neural networks (CNNs) to create a solution to the “detection and classification of acoustic scenes and events” (DCASE) 2016 challenge. Applications of CNNs for audio-related tasks is becoming more common in a range of other applications. This work intends to re-produce the results generated by Valenti et al and propose an improvement that could be made to the system.

II. Related Work

Further on from the work of Valenti et al in 2016, there has been more submissions to DCASE challenges for the later years. The work from 2017 by Mun et al [5] uses generative adversarial networks (GANs) in order to generate additional training data in order to improve performance of the model. The authors propose the use of Support Vector Machine (SVM) hyper planes for each class as a reference for selecting samples generated by the GAN. This technique allowed them to achieve an accuracy of 83.2%.

In 2018, the work of Sakashita [6] adaptively divides the produced spectrograms and learns multiple neural networks to produce a result. The 2018 data set audio was recorded using a binaural microphone so spectrograms were produced from the binaural audio and mono audio to produce more input data. This ensemble method produced a model accuracy of 81% against the evaluation dataset.

III. Dataset

The data has been provided in the form of log-mel spectrograms. The data is stored as numpy arrays that represent the produced spectrograms. There are 1170 segments in the development training dataset. For non-full training, following Valenti et al, we will use 880 training segments for the train test split leaving 270 for validation. For full training, we will use the evaluation dataset of length 390.

The files are labeled by their number and their segment start and end time. The samples are already split into 30 second clips from the original sound recording. There are 15 classes that the data fall in to. They are as follows: beach, bus, cafe/restaurant, car, city center, forest path, grocery store, home, library, metro station, office, park, residential area, train and tram.

IV. Input

The authors chose to preprocess the data by converting the audio files into log-mel spectrograms. The log-mel spectrogram is calculated by applying a short-time Fourier transform (STFT) over windows of 40 ms of audio with a 50% overlap Hamming windowing. The absolute value of each bin is then squared and a 60-band mel-scale filter bank is applied. Finally, the logarithmic conversion of the mel energies is computed. After extraction each bin is normalized by subtracting its mean and dividing by its standard deviation. Each normalized spectrogram is then split into shorter spectrogram known as segments as mentioned in Section III.

Figure 1 shows an example spectrogram produced from the audio of a file with the label of ‘cafe/restarunt’. Time is represented on the x-axis and frequency on the y-axis. The brighter colours show a louder volume and the darker colours show a quieter volume.

Figure 2 shows another example spectrogram, this time produced from the audio of a file with the label of ‘metro

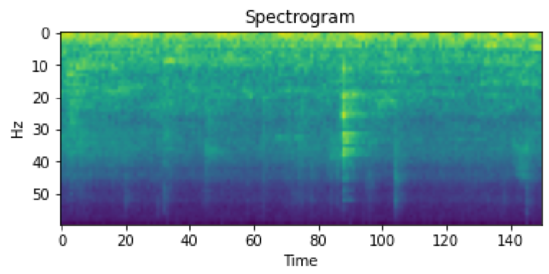

Fig. 1. Spectrogram Example of Ground Truth: cafe/restaurant

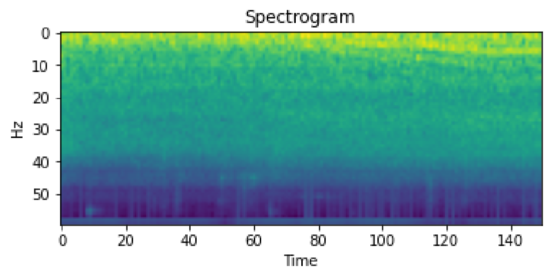

Fig. 2. Spectrogram Example of Ground Truth: metro station

station'. By comparing these two spectrograms together you can see how they differ, showing the audio differences between the two environments.

V. CNN Architecture

TABLE I
CNN Architecture Blocks

Block Name	Configuration
Input	log-mel spectrograms, clip length 3, batch size 64
Conv2d 1	128, 5x5 Kernels
Batch Norm 1	Default
ReLU	
MaxPool2d 1	5x5 Kernel with 5x5 Stride
Conv2d 2	256, 5x5 Kernels
Batch Norm 2	Default
ReLU	
AdaptiveMaxPool2d 1	Output size 4x1
Fully Connected 1	Size 25600x15

The architecture designed by Valenti et al uses two convolutional layers and one fully-connected layer. Each block is described in Table I.

Firstly, a convolution is performed over the input spectrogram with 128 kernels with a receptive field of 5×5 with a depth and stride of 1 in both dimensions. The output feature maps are then batch normalized and passed through the rectifier function for activation. They are then passed through a max-pooling layer operating with 5×5 non-overlapping squares. The second convolutional layer is identical to the first but uses 256 kernels rather than 128 with batch normalization and the same rectifier function for activation. Together, these are known as rectified linear units (RELUs). Next, further subsampling is carried out in order to cause "destruction" of the time axis. A max-pooling layer is used that operates over the entire sequence length and only over 4 non-overlapping frequency bands on the frequency axis. The classification involves 15 different classes so the final layer is a softmax layer containing 15 fully connected neurons.

The system implemented by Valenti et al uses the Keras library for Python (we will be using PyTorch). For training, the loss function used is categorical cross-entropy and adaptive momentum (adam) [7] is used for the optimization algorithm.

VI. Implementation Details

The described CNN is implemented using the PyTorch Python library. To start, the input data is provided through the use of a given helper class `DCASE`. The log-mel spectrograms are provided in the form of numpy arrays which are then loaded into the main program using a PyTorch Dataloader. Prior to this, a train and test split of the development data is carried out to ensure there is an acceptable distribution of inputs per class in each set. This train and test split are created separate to the network before execution starts in order to preserve the same split for future runs. At the beginning of execution, the loss function and optimizer are defined using the built in classes of PyTorch. The `CrossEntropyLoss` is used for the criterion along with the `Adam` optimizer to match those used in the original paper. They are setup using their default values. This gives a learning rate of $1e^{-3}$ for the adaptive momentum optimizer.

The model is created by extending the `nn.Module` class, in our implementation this is named `CNN`. The initialisation function and forward pass functions are overridden in the `CNN` class. The initialiser contains the code to set up all of the blocks that are part of the architecture. This includes the convolutional layers, batch normalisation and max-pooling layers. The sizes of these layers can be found in Table I. Layers are also initialised individually in this part of the model using a static function. The forward pass function defines the behavior of the data that will be passed through the model. It connects the layers in the correct order for processing. This function is where we also apply the rectifier activation function. As the provided data is given in the size `[batch_size, num_clips, height, width]`, we also need to reshape the input data into the size `[batch_size * num_clips, height, width]` so that the CNN layers receive the data in the correct form. This is done using the `torch.view()` function before the data is passed through the network. We also use an adaptive max pooling layer to represent the papers "destruction" of the time axis. At the end of the forward pass, the data is reshaped back into the original size and an average is taken across the `num_clips` axis in order to produce the classification for the whole segment.

For training, a `Trainer` class is created. This contains the steps to initialise the model trainer and functions for training, validation and other helpers such as logging. The `train()`

function contains the code to train the model. It contains a for loop that steps through each epoch and within each epoch step, a second for loop iterates over each batch. Within the batch loop, the data is passed forward through the model using the `model.forward()` function we have written, and the loss is calculated using the given criterion defined at the beginning. After this step, the backward pass is then carried out to update the weights. Accuracy and loss are also calculated at this step and are logged or printed if the step frequency is met. After each batch, the model is also validated depending on the validation frequency.

The `validation()` function carries out no optimisation, it only calculates the model predictions with the test data and produces values for accuracy and loss at a given epoch. It first switches the model to evaluation mode before preceding. In non-full training, the validation function also saves the parameters of the model if the accuracy has improved since the last validation step. The function also logs the newly calculated accuracy and loss values.

Once the training has completed, the model is validated again against the test set and the accuracy result is outputted along with a confusion matrix for the given test data. The `createConfusionMatrix` function uses all the test data to create a confusion matrix based on the predictions and ground truth for each input and saves the image to the disk.

VII. Replicating Quantitative Results

Replicating results from a paper rarely produce identical outcomes. It is especially difficult without access to the code but an attempt is made.

TABLE II
Replicated Results

System Head	Seq len(s)	Accuracy	
		non-full	*full*
Two-layer CNN (log-mel)	3	82.6%	86.4%

The model produces an accuracy of 82.6% during non-full training and 86.4% when training using the full development dataset. Figure 3 also shows the confusion matrix for the model. The non-full training accuracy is within 2% of the target value. The full training accuracy is within 1% of the target value.

VIII. Training Curves

Firstly, Figure 4 and Figure 5 show the curves that were produced when running non-full training on the model.

During full model training, the curves in Figure 6 and Figure 7 were produced.

It can be seen from the training curves from both non-full and full training that there is some slight overfitting of the training data due to the gap between the train and test curve. The results show that this overfititng is worse during non-full training. This is expected due to the little data that is available. It can also be seen that there is some oscillation of test accuracy around a stable value showing some system convergence.

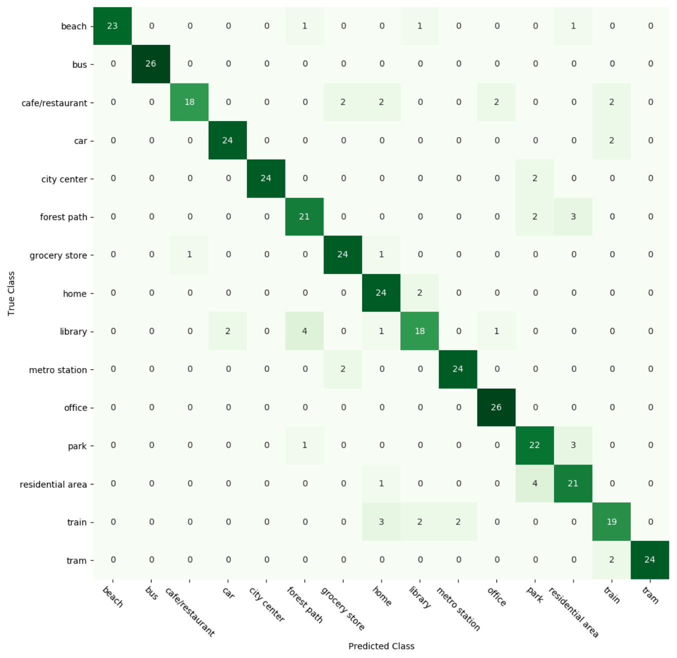

Fig. 3. Confusion Matrix of Network Ouptut

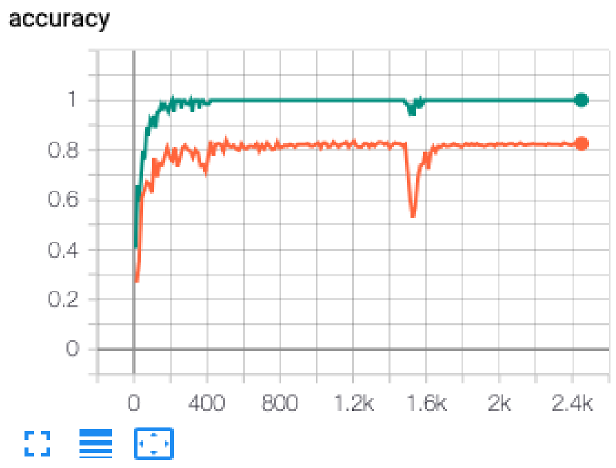

Fig. 4. Full Training Accuracy Curve - Green Line = Train Curve, Orange Line = Test Curve

IX. Qualitative Results

It can be seen from the confusion matrix in Figure 3 that the model performs well on certain classes but not as well on others. For example, the model does well in predicting audio from the 'bus' class. All 26 evaluation samples were classified successfully. Figure n shows an example spectrogram taken from the 'bus' class.

There were some classes where the model struggled with classification. For example, Figure 9 shows a spectrogram from the class 'cafe/restaurant' that was miss-classified as a 'grocery store'.

Another example where there was an unsuccessful classi-

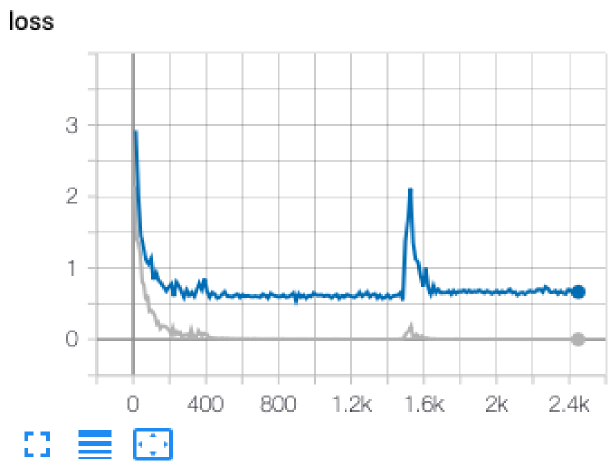

Fig. 5. Full Training Loss Curve - Blue Line = Train Curve, Grey Line = Test Curve

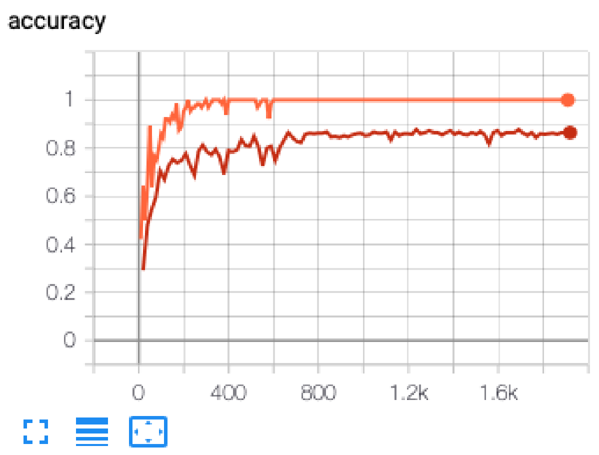

Fig. 6. Full Training Accuracy Curve - Orange Line = Train Curve, Red Line = Test Curve

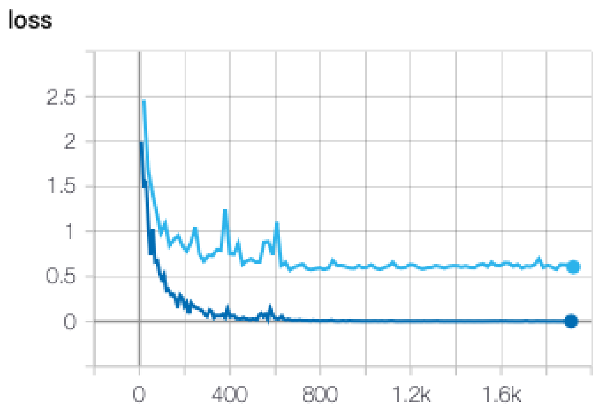

Fig. 7. Full Training Loss Curve - Dark Blue Line = Train Curve, Light Blue Line = Test Curve

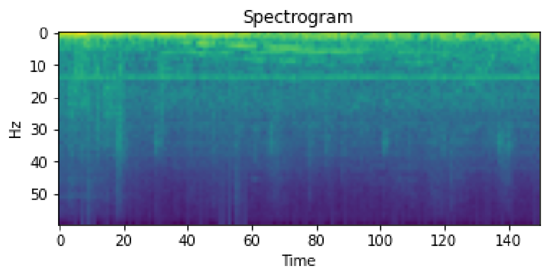

Fig. 8. Spectrogram for Sample from 'bus' class - Successfully classified as 'bus'

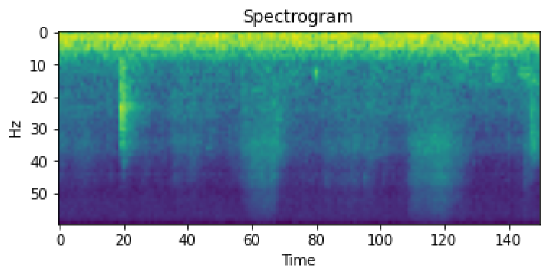

Fig. 9. Spectrogram for Sample from 'cafe/restaurant' class - Miss-classified as 'grocery store'

fication was within the 'library' class. Figure 10 shows an example spectrogram from this class that was miss-classified as a 'forest path'.

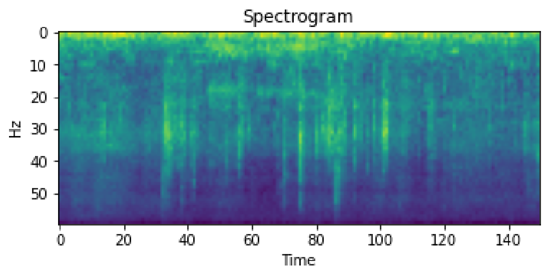

Fig. 10. Spectrogram for Sample from 'library' class - Miss-classified as 'forest path'

The unsuccessful results from these classes could be due to a number of causes. It can be seen from the spectrograms that they contain more variability in sound than the 'bus' which may make it more difficult for the model to extract features that can separate those classes from others. It can also be observed that there are some louder low frequency components in both the 'cafe/restaurant' and 'library' spectrograms compared to that of the 'bus' this may be affecting the accuracy of the classifier.

X. Improvements

There are many different ways to improve any existing CNN. Data augmentation is a method where transforms are applied to existing data but their labels are preserved. Online augmentation is most commonly carried out when working with image data as transforms such as flipping or rotation can be easily carried out. In our case, these common image transforms are unsuitable for the task but other transforms suitable for spectrograms do exist. SpecAugment [8] is the work of Park et al from Google who have discovered suitable augmentation for working with spectrograms. The authors found that time and frequency masks improved performance when working with spectrograms in deep neural networks.

I will propose the implementation of frequency and time masking as an augmentation technique to improve the performance of the model. These augmentations are easy to implement. They can be applied as a transform to the dataset loader. Two custom transform classes were written, `FrequencyMask` and `TimeMask`. They can then be combined using the PyTorch `transforms.Compose` function and applied to the dataset. This required some additional modifications to the dataset loading class in `dataset.py`

The use of these augmentations unfortunately does not improve the performance of the model. Experimenting with different hyper parameters for the augmentation ended in no increase in accuracy. It is apparent that there is not much room for improvement within this model without any major adjustments to architecture or by the introduction of other technologies such as generative adversial networks to create more input data.

XI. Conclusions and Future Work

In conclusion, this work has described and reproduced the work of Valenti et al on the 2016 DCASE dataset. We have successfully re-implemented the CNN that was used in the original paper, achieving an accuracy of 86.4% when running full training. This report has also detailed the implementation of the CNN using the PyTorch Python library, allowing for easy reproduction if needed.

Future work could look into the use of a GAN in order to produce more training data for the model as this is an obvious limitation to the network. Using a GAN will allow the model to be trained using more data which will in term improve its performance. Training an ensemble of networks is another addition that could allow the model to perform better and generalise to more data.

References

[1] B.Schilit, N.Adams, and R.Want, “Context-aware computing applications,” in Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on. IEEE, 1994, pp. 85–90.

[2] Y.Xu, W.J.Li ,and K.K.Lee,Intelligent wearable interfaces. John Wiley & Sons, 2008.

[3] S.Chu, S.Narayanan, C.-C.J.Kuo, and M.J.Mataric, “Where am I? Scene recognition for mobile robots using audio features,” in 2006 IEEE International Conference on Multimedia and Expo. IEEE, 2006, pp. 885–888.

[4] Valenti, M., Diment, A., Parascandolo, G., Squartini, S., & Virtanen, T. (2016). DCASE 2016 ACOUSTIC SCENE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS.

[5] Mun, S., Park, S., Han, D.K., & Ko, H. (2017). GENERATIVE ADVERSARIAL NETWORK BASED ACOUSTIC SCENE TRAINING SET AUGMENTATION AND SELECTION USING SVM HYPERPLANE.

[6] Sakashita, Y. (2018). Detection and Classification of Acoustic Scenes and Events 2018 Challenge ACOUSTIC SCENE CLASSIFICATION BY ENSEMBLE OF SPECTROGRAMS BASED ON ADAPTIVE TEMPORAL DIVISIONS Technical Report.

[7] D.Kingma and J.Ba ,“Adam:A method for stochastic optimization,” arXiv preprint arXiv:1412.6980, 2014.

[8] Park, D. S., “SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition”, arXiv e-prints, 2019.