ARS Task Report

Thevin Degamboda
tdegamboda_5@live.com

June 22, 2021

Version Information

The following indicates the versions of the software used to produce the code below and all source code in the repository:

- Python version 3.7.4
- Tensorflow version 1.15
- h5py version 2.10.0

Further information on how to navigate the repository can be found in the `README.md`.

Question 1

By taking advantage of Tensorflow's implementation of the Keras API we can directly refer to [8], make improvements to arbitrary code sections, and implement the SwishNet using `tf.keras`. My implementation can be found in the `Q1` directory of the repository along with the Keras implementation from [8]. By seeding the random generator I tested my implementation against the results of [8]. The following code was used in order test both implementations,

```
np.random.seed(2021)
net = SwishNet(input_shape=(16, 20), classes=2)
print(net.predict(np.random.randn(2, 16, 20, )))
```

Results of the Keras implementation,

```
[[0.4833677  0.51663226]
 [0.47845703 0.5215429 ]]
```

Results of my implementation,

```
[[0.4833677  0.51663226]
 [0.47845703 0.5215429 ]]
```

These results can be further confirmed from the outputs in the notebooks of the `Q1` directory.

Question 2

The following is a low-level overview of the steps taken to preprocess the MUSAN dataset, as described in [3]

1. **Silence Removal**
    - The audio was first processed into a one-dimensional time series with a sampling rate of 16kHz using the `librosa` library
    - Then over a window of 250ms we compute the power using the logarithmic scale
    - After considering a series of threshold values, we set the threshold value to 40dB. This level is used to describe a quiet library or a close whisper and hence this threshold value seems appropriate in order to extract meaningful features
2. **Loudness Equalisation**
    - Note in [3], Hussain *et al.* mentions equalising loudness over a window length of 250ms however, this can be interpreted as normalising the loudness over a window of 250ms
3. **Segmentation**
    - Resulting signals were segmented to form files of 0.5s, 1s, and 2s clips with 50% overlap
    - This can be conveniently done by computing how many frames to consider for each clip i.e. ,

$$\text{Number of frames per } x \text{ second clip} = x \times \text{Sample Rate}$$

    - Using this we segment our cleaned signal
4. **Feature Extraction**
    - Clips were framed into 25ms frames with 15ms overlap in order to extract 20 MFC coefficients from 32 frequency bands
    - Note in order to specify the number of frequency bands we use the `mfcc` method from the `python_speech_features` library

The preceding steps were compiled into a pipeline which took the file path to the MUSAN corpus as an input and returned an .h5 file with the MFCC features and encoded labels of each clip along with a .txt file of the ordered file names.

Table 1: Overall and Speech/Non-Speech (SNS) Classification Accuracy for Clips of Different Lengths

Clip Length	Overall (%)	SNS (%)
0.5s	97.74	99.05
1.0s	98.43	99.49
2.0s	99.02	99.77

Question 3

For training the following strategy was implemented, as described in [3],

- The MUSAN corpus was randomly divided into training, validation and test sets where 65% of files was assigned to the train, 10% was assigned to the validation and the remaining 25% was assigned to the test set
- Given that we are solving a classification task, the loss function for this model is Categorical Cross-Entropy
- The optimizer used to train all models was the Adam optimizer
- Cosine annealing and warm restarts were implemented as described in [5]. With reference to [4], this was implemented in the form of a Tensorflow Callback that could simply be initialised at training and implements the learning rate schedule.
- The initial learning rate was set to 0.0005 and the minimum learning rate was set to 0.00001
- The model was trained for 120 epochs with a batch size of 128 on a dual-core CPU
- Outside the scope of [3], in order to speed up training we apply an early stopping condition where if the validation accuracy does not increase over 20 epochs then we terminate training. Furthermore, the parameters that produced the highest validation accuracy over training are loaded to the model to evaluate the trained model on the test set.

We observed a relatively "normal" and expected plot of the loss and accuracy over training where we observe a steep gradient at the beginning of training followed by plateauing behaviour. It is interesting to note certain cyclic spikes during training which can be associated with the cosine annealing technique used to compute the learning rate on each epoch. The following plots visualise the loss and accuracy of both the training set and validation set over training on the features of the 2 second clips.

(a) Loss (b) Accuracy

Figure 1: Training on 2 second clips

Evaluation on the test set of the MUSAN corpus produces results that are within an acceptable interval of the results reported in [3], as observed in Table 1, hence I can suggest with high confidence that the implementation I have used in order to produce the results in this report satisfies the experimental setup described in [3]. For a sanity check, i.e. ensure accuracy is not overly-inflated due to underlying issues in the distribution of the dataset and labels, we also plot the normalised confusion matrix which further confirms the results of Hussein *et al.* . As we observe from figure 2 the model continues to have difficulty with distinguishing noise and music, which is expected as most humans can often find it difficult to distinguish the two in such small clips.

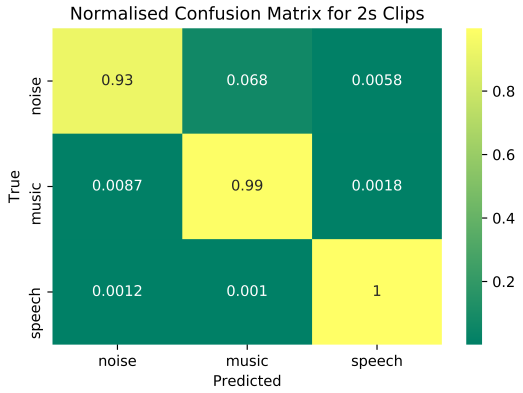

Figure 2: Confusion matrix from evaluation on the 2s clips test set of the MUSAN corpus

Question 4

To prepare the GTZAN corpus, we remove background noise using the Log MMSE method. Following this step, we preprocess the cleaned corpus as described in the preprocessing pipeline used for the MUSAN corpus. For evaluation, we take the following approach,

- The GTZAN dataset is split, where we take 25% of dataset for fine-tuning and the rest is kept for testing

- The model is trained for 50 epochs with a learning rate of 0.0005. We set the learning rate low in order to avoid the model from overfitting on this dataset.
- Note that the GTZAN dataset only contains data for two of the labels, music and speech, that the initial model was trained on

The following summarises the classification results of the evaluation of my implementation of the SwishNet model,

Table 2: GTZAN Music/Speech Evaluation for Clips of Different Lengths

Clip Lengths	**Overall Accuracy (%)**	**Speech Recall (%)**	**Music Recall (%)**	**Average Recall (%)**	**F1 Score (%)**
0.5s	91.61	89.14	92.81	91.00	90.47
1.0s	93.46	92.38	93.07	92.73	92.98
2.0s	95.68	96.24	94.33	95.29	95.04

Given that we have achieved results within 3% of those reported in the research paper for 2 second clips, we will continue discussing our SwishNet model without training a separate Neural Network. We can assume that due to the use of an early stopping condition in the initial training of the SwishNet, our models have been trained for a shorter time hence less generalised which can directly impact performance for the fine-tuning task. With more time and resources, it would be worth training the model on the full MUSAN corpus for the exact number of epochs quoted in [3], and once again evaluating it on the GTZAN dataset to observe the models ability to generalise with more data and time for training.

Question 5

Hussain *et al.* present a neural network which achieves high accuracy on audio classification, with proven robustness on a different corpus, while being fast, lightweight and memory-efficient. In particular, SwishNet is a 1D convolutional neural network, i.e. convolutions are carried out along the time axis, which is capable of taking frame-wise MFCC features as an input and classifying each input signal into one label. By carefully designing a network of 1D convolutions Hussein *et al.* are able to build a 1D CNN that requires only a fraction of the memory needed for a 2D CNN. We can summarise the main features of the SwishNet as follows,

- The model is fast on a CPU and consumes a low amount of memory
- The model achieves high accuracy on clip-classification and frame-wise segmentation tasks and even higher accuracy in speech/non-speech discrimination tasks
- Such tasks are commonly tackled in the area of Voice Activity Detection (VAD) algorithms. Hence we can note the pragmatic potential of this network as an embedded layer in a VAD system

- To further establish the model's ability to also improve from transfer learning, we observe strong results from the distillation strategy applied to the MobileNet which was pretrained on ImageNet.

Since this paper [3] was published in 2018, there have not been any further developments on the SwishNet or references to it in recent work in Voice-Activity Detection algorithms or music/speech classification tasks.

Overall, this paper presents both an end-to-end framework as well as a tool that can be used amongst other systems in order to achieve strong performance in audio discrimination tasks without consuming large amounts of memory or processing power. On further analysis into the experimental setup and approach of this model I believe it is worth considering further experiments to better suit the domain of the data. There are many approaches to handling sequential data and in recent years, we have observed classical models, such as HMMs and GMMs, be outperformed by simple feed-forward neural networks (FNNs). However, the Recurrent Neural Networks (RNNs) have proven to be very powerful with sequential data as they use weight sharing in order to store past information in later time slices. In [2] Hughes *et al.* presents a novel RNN model for voice activity detection. Their RNN model, in which nodes compute quadratic polynomials as opposed to a weighted sum and non-linear activations as seen in FNNs, can outperform larger GMM-based systems on VAD tasks. Thus, it would be worth applying an RNN model, such as an LSTM or GRU, to the tasks described in this paper [3]. Furthermore, Choi *et al.* [1] applied their convolutional recurrent neural network, a stacked hybrid model consisting of multiple CNN and RNN layers, to a series of music tagging tasks. Using t-SNE Nasrullah *et al.*[6] visualise the audio representations learnt by the model at the bottleneck layer in order to further enforce the models ability to learn rich semantic representations of the audio input. However, we note that this is a much heavier network hence falls short on the benefits of speed and memory-efficiency provided by [3].

Likewise, another growing technique used in large-scale deep learning frameworks that have proven to improve performance on shallow networks is transfer learning. In [7], Palanisamy *et al.* shows that considering standard models, such as Inception, ResNet, DenseNet pre-trained on ImageNet, as a baseline model to distill features down to a shallow network produced state-of-the-art results in audio classification on two datasets. Observing the use of other pre-trained models, besides MobileNet, in order to train our SwishNet would be worth considering as the MobileNet consistently performed the best in [3].

In conclusion, the research paper presented in this task satisfied a very specific use case: the need for a model that is fast, light, and can be easily integrated into existing systems. It does not necessarily dive into considering specific features of audio and how humans perceive audio. That being said, a model like this can easily be deployed in Tensorflow Lite and embedded into a hearing aid where discriminating speech and non-speech allows the device to amplify speech signals for the user. The ease of mobility with such a model can be substantially beneficial towards app developers and other applied scientists.

References

[1] Keunwoo Choi et al. "Convolutional Recurrent Neural Networks for Music Classification". In: *CoRR* abs/1609.04243 (2016). arXiv: 1609.04243. URL: http://arxiv.org/abs/1609.04243.

[2] Thad Hughes and Keir Mierle. "Recurrent neural networks for voice activity detection". In: (2013).

[3] Md. Shamim Hussain and Mohammad Ariful Haque. "SwishNet: A Fast Convolutional Neural Network for Speech, Music and Noise Classification and Segmentation". In: *CoRR* abs/1812.00149 (2018). arXiv: 1812.00149. URL: http://arxiv.org/abs/1812.00149.

[4] *Learning Rate Schedulers*. URL: https://github.com/Mr-TalhaIlyas/Learning-Rate-Schedulers-Packege-Tensorflow-PyTorch-Keras.

[5] Ilya Loshchilov and Frank Hutter. "SGDR: Stochastic Gradient Descent with Restarts". In: *CoRR* abs/1608.03983 (2016). arXiv: 1608.03983. URL: http://arxiv.org/abs/1608.03983.

[6] Zain Nasrullah and Yue Zhao. "Music Artist Classification with Convolutional Recurrent Neural Networks". In: *CoRR* abs/1901.04555 (2019). arXiv: 1901.04555. URL: http://arxiv.org/abs/1901.04555.

[7] Kamalesh Palanisamy, Dipika Singhania, and Angela Yao. "Rethinking CNN Models for Audio Classification". In: *CoRR* abs/2007.11154 (2020). arXiv: 2007.11154. URL: https://arxiv.org/abs/2007.11154.

[8] *Swishnet Keras Implementation*. URL: https://github.com/i7p9h9/swishnet.